JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Jan Schlüter

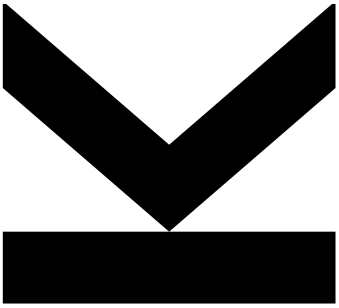
Submitted at
Department of Computational Perception

Supervisor and
First Examiner
Gerhard Widmer

Second Examiner
Simon Dixon

July 2017

Deep Learning for Event Detection, Sequence Labelling and Similarity Estimation in Music Signals



Doctoral Thesis
to obtain the academic degree of
Doktor der technischen Wissenschaften
in the Doctoral Program
Technische Wissenschaften

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Any sufficiently advanced technology is indistinguishable from magic.
— *Arthur C. Clarke, 1973*

Statutory Declaration

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references. This printed thesis is identical with the electronic version submitted.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Dissertation ist mit dem elektronisch übermittelten Textdokument identisch.

Deep learning is not magic.
— *Yoshua Bengio, 2014*

Abstract

When listening to music, some humans can easily recognize which instruments play at what time or when a new musical segment starts, but cannot describe exactly how they do this. To automatically describe particular aspects of a music piece – be it for an academic interest in emulating human perception, or for practical applications –, we can thus not directly replicate the steps taken by a human. We can, however, exploit that humans can easily annotate examples, and optimize a generic function to reproduce these annotations. In this thesis, I explore solving different music perception tasks with deep learning, a recent branch of machine learning that optimizes functions of many stacked nonlinear operations – referred to as deep neural networks – and promises to obtain better results or require less domain knowledge than more traditional techniques.

In particular, I employ fully-connected neural networks for music and speech detection and to accelerate music similarity measures, and convolutional neural networks for detecting note onsets, musical segment boundaries and singing voice. In doing so, I evaluate both how well and in what way the networks solve the respective tasks. Using the example of singing voice detection, I additionally develop data augmentation methods to learn from only few annotated music pieces, and a recipe to obtain temporally accurate predictions from inaccurate training examples.

The results of my work surpass the previous state of the art in all the tasks considered. The learned solutions are similar to existing hand-designed approaches, but are more extensively optimized than possible by hand. Both indicates that the same methods could also yield substantial improvements for other machine listening problems. The self-contained description of my work – including a thorough introduction to all relevant deep learning and signal processing techniques – and my contributions to several open-source software projects shall help other researchers and practitioners to accomplish exactly that.

In conclusion, this thesis both advances the state of the art in five concrete applications, and, on a higher level, participates in the ongoing democratization of deep learning.

Zusammenfassung

Einige Menschen können beim Hören einer Musikaufnahme sehr leicht erkennen, wann welche Instrumente spielen oder ein neuer Abschnitt beginnt, aber nicht genau erklären, wie sie dies tun. Um ein Musikstück automatisch nach bestimmten Kriterien zu beschreiben – sei es aus dem akademischen Interesse heraus, menschliche Wahrnehmung nachzuahmen, oder für praktische Anwendungen –, können wir daher nicht direkt die Schritte kopieren, die ein Mensch dazu befolgt. Wir können allerdings ausnutzen, dass Menschen sehr leicht Beispiele annotieren können, und eine generische Funktion dahingehend optimieren, diese Annotationen zu reproduzieren. In dieser Dissertation untersuche ich die Lösung verschiedener Aufgaben der Musikwahrnehmung mittels Deep Learning, eines jungen Teilgebiets des Maschinellen Lernens, das Funktionen aus einer Abfolge an vielen nichtlinearen Operationen – sogenannte tiefe neuronale Netze – optimiert und dabei verspricht, bessere Ergebnisse zu erzielen oder weniger Domänenwissen vorauszusetzen als herkömmlichere Methoden.

Konkret setze ich voll verbundene neuronale Netze ein, um Sprache und Musik zu detektieren sowie Musikähnlichkeitsmaße zu beschleunigen, und faltungsbasierte neuronale Netze, um Notenanfänge, Musikabschnitte und Gesang zu finden. Dabei erfasse ich nicht nur wie gut, sondern auch auf welche Weise die neuronalen Netze die Aufgaben lösen. Am Beispiel der Gesangsdetektion entwickele ich außerdem Methoden der Datenaugmentierung, um aus wenigen Musikstücken zu lernen, und ein Rezept, um aus zeitlich ungenau annotierten Trainingsbeispielen genaue Vorhersagen zu erzielen.

Die Ergebnisse meiner Arbeit übertreffen den bisherigen Stand der Technik in allen untersuchten Aufgaben. Die gelernten Lösungen ähneln existierenden manuell entworfenen Ansätzen; ihr Vorteil liegt in einer viel weitgehenderen Optimierung, als sie von Hand möglich wäre. Beides lässt erwarten, dass die gleichen Methoden auch für weitere Probleme des maschinellen Hörens wesentliche Verbesserungen erzielen könnten. Die abgeschlossene Beschreibung meiner Arbeit – inklusive einer umfassenden Einführung in alle verwendeten Methoden des Deep Learning und der Signalverarbeitung – sowie meine Beiträge zu mehreren quelloffenen Softwareprojekten sollen anderen ForscherInnen oder PraktikerInnen eben dies erleichtern.

Zusammengefasst verbessert diese Dissertation den Stand der Technik in fünf konkreten Anwendungen, und leistet darüber hinaus einen Beitrag zur fortschreitenden Demokratisierung von Deep Learning.

Acknowledgements

While a few pages ago I declared that this thesis is my own unaided work, I am grateful and indebted to many people who influenced and supported me on my journey leading towards and through my doctoral studies.

First and foremost, I would like to thank my supervisor *Gerhard Widmer* for all his trust over the years, for virtually unlimited freedom in pursuing my research and side projects, for interesting industrial applications to work on, and for a (short) opportunity to enjoy teaching again. Secondly, I warmly thank my second examiner *Simon Dixon* for his time and effort to review the results of my research.

This thesis is a logical continuation of the route taken in my master's studies. For this, I am still grateful to *Breno Faria*, for encouraging me to do machine learning instead of symbolic AI and for giving me a kick-start in Python and numpy, and to *Christian Osendorfer*, for his machine learning lecture and project and for initiating my quest to apply deep learning to music. Without you two, I would have discovered deep learning much later and missed all the fun.

The work described in this thesis was mostly carried out at the Austrian Research Institute for Artificial Intelligence (OFAI) in Vienna, with occasional collaborations with the Department of Computational Perception (CP) at the Johannes Kepler University (JKU) in Linz, and I am grateful to the many great colleagues both at OFAI and CP/JKU: *Thomas Grill* for fruitful and enjoyable collaboration both on a grant proposal led by Gerhard Widmer and the ensuing project, *Dominik Schnitzer* for discussions on music similarity and hubs, and for teaching me the necessary skills left to know to administrate our group's code repository server, which has slowly turned into a GPU machine since, *Arthur Flexer* for his unbreakable good mood, a second successful grant proposal I am excited to work on soon, and for donating a much-needed second GPU server from his project, *Sebastian Böck* for his data and guidance when I set to beat him in onset detection using CNNs, as well as for invaluable travel experiences, *Karen Ullrich* for her unshakable belief that my onset detection CNN would also work for structural segmentation, and for helping in piecing together the second GPU server, *Constantin Marsch* and *Paolo Petta*, without whom OFAI would have lost its internet connectivity and all major servers many times, *Karin Vorsteher*, *Inge Hauer*, and *Gudrun Schuchmann*, the backbones of all research carried out at OFAI, and *Robert Trappl* for founding and running the institute. Furthermore, I thank all of the above and – in Mersenne-twisted order – *Maarten Grachten, Florian Krebs, Harald Frostel, Reinhard Sonnleitner, Matthias Dorfer, Peter Knees, Rainer Kelz, Stefan Lattner, Richard Vogl, Laura Bishop, Filip Korzeniowski, Martin Gasser, Andreas Arzt, Hamid Eghbal-Zadeh, Carlos Cancino, Andreu Vall, Roman Feldbauer, Klaus Seyerlehner, Bernhard Lehner, André Holzapfel, Markus Schedl* and *Thassilo Gadermaier* for shared thoughts, shared meals, shared evenings, shared train rides or flights, shared apartments, and shared fun.

My work was also influenced by ideas I caught up or discussed at conferences, and I am especially grateful to all the folks making ISMIR so pleasant and inspiring to attend over the years, again in randomized order and hopelessly incomplete: *Colin Raffel, Bob Sturm, Sander Dieleman, Eric Humphrey, Eric Battenberg, Rachel Bittner, Erik Schmidt, Masataka Goto, Simon Dixon, Philippe Hamel, Juan Pablo Bello, Xavier Serra, Matthias Mauch, Fabien Gouyon, Geoffroy Peeters, Siddhart Sigtia, Oriol Nieto, Meinard Müller, Stephen Downie* and *Brian McFee*.

Carrying out the experiments in this thesis would have been impossible without a number of open source projects, so I would like to thank the authors and coauthors of all the software I relied on, in especially: *Python, IPython, numpy, scipy, matplotlib* for providing a solid and open scientific software stack, *Marc'Aurelio Ranzato* for his mcRBM implementation, which was my start into deep learning and GPU programming, *Vladimir Mnih* for *cudamat*, which the mcRBM implementation and my first neural networks were based on, *Mohammad Norouzi* for publishing his HDML implementation, all the contributors to *Theano*, which has made my work incredibly easier, especially the tireless maintainers *Frédéric Bastien, Pascal Lamblin* and *Arnaud Bergeron* for their willingness and help to include my improvements, and finally *Sander Dieleman* for starting the *Lasagne* project and assembling a talented team it was very pleasant to work with: *Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Eric Battenberg, Aäron van den Oord* (ordered by number of commits), as well as over 50 other contributors.

Finally, I thank my friends and all the lindy hoppers in Vienna for good spirits, and my family for enabling me to follow my curiosity wherever it lead me.

The research for this thesis was funded by the Austrian Science Fund (FWF), project Z159 ("Wittgenstein Award"), and by the FWF and the Federal Ministry for Transport, Innovation & Technology (BMVIT), project TRP 307-N23. I also gratefully acknowledge the support of NVIDIA Corporation with the donation of two Tesla K40 GPUs used for this research.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Music recordings are very complex audio signals often composed from many different sound sources and richly structured both in terms of occurring pitches and over time. Nevertheless, when some humans listen to music, they can effortlessly register a lot of details: When does somebody start or stop singing? Does the voice sound female or male? Which other instruments are present? When exactly do they play their notes? How fast is the music piece? Where are the beats and bars? When does the rhythm, instrumentation or melody change? However, despite their ease in capturing such cues, they have a hard time explaining exactly *how* they accomplish this.

When attempting to answer similar questions about a music recording with a computer – be it for an academic interest in replicating aspects of human perception, or for practical applications – we can thus not directly formalize the process used by humans and write a computer program following the same steps. Instead, we can try to look at a set of recordings and figure out an algorithm that produces answers similar to those given by humans, exploiting our knowledge on how music is produced and structured. For some tasks, this is relatively easy: To detect the note onsets in a piano solo recording, it can be sufficient to locate sudden increases of loudness caused by the hammer hitting the strings, achievable with basic signal processing methods. Other tasks such as detecting a change of instrumentation require substantially more engineering. Furthermore, even for seemingly simple problems, it is hard to achieve human-level performance for complex polyphonic music pieces and across a wide range of instruments and playing styles with a hand-designed algorithm.

This is the ideal scenario for machine learning: Since humans can easily solve the task, we can gather a lot of example pairs of recordings and expected descriptions, and then automatically optimize a generic computer program to best approximate the human annotations. The problem of handling the complexity and variability of music then becomes a problem of finding a diverse enough set of training examples to constrain optimization to converge to a robust solution.

Now of course, machine learning is not devoid of difficult design decisions and engineering problems either. At the very least, we need to define the structure of the generic program to be optimized, a metric for measuring its success and a method to optimize the program to maximize this metric in a computationally efficient way. Furthermore, machine learning is not almighty: Depending on the complexity of the task and the machine learning method, we may not be able to replace all of the solution by an automatically optimized program, but still need to implement a substantial part of it manually to constrain the search space, such as by computing domain-specific input features or post-processing the output. Thus, the application of machine learning is always a tradeoff: Any reduction in the amount of required domain knowledge and hand-design for an engineered solution must be made up for by machine learning expertise and training data. However, the hope is that machine learning expertise is more universally applicable than domain knowledge, making it easier to solve new tasks, and optimizing larger parts of a solution from data will ultimately lead to better solutions.

In this thesis, I explore solving different music perception tasks with deep learning. As a branch of machine learning, it investigates a particular area in the design space of machine learning algorithms in which the generic program to be optimized is based on potentially long chains of nonlinear operations termed deep neural networks. When I started this thesis, this approach showed some initial success for image analysis and speech recognition, both in reducing the amount of required hand-design and in improving results, but had hardly been applied in the music domain. Filling this gap seemed promising for advancing the field of music information retrieval, but also for learning about deep learning and subsequently spreading the expertise and domain knowledge required to apply it to audio signals.

1.2 Contributions

The scientific contributions from this endeavour are manifold:

(1) I show how to adapt and apply deep learning methods to two sequence labelling and two event detection tasks, as well as to speed up music similarity measures. The solutions may serve as proven examples for tackling related tasks.

(2) I evaluate how well deep learning performs compared to existing approaches: it improves results over the previous state of the art in all five tasks. This provides benefits for practical applications, and a point of reference for future work.

(3) I investigate what the learned solutions do, showing that they do not find surprising new ways to solve a task, but are more extensively optimized than possible by hand. This improves confidence in the learned solutions over treating them as black boxes.

(4) I provide a self-contained description of my work, including a thorough introduction to the relevant deep learning and signal processing techniques I learned on the way. This may allow other researchers and practitioners to quickly adopt deep learning for their tasks.

(5) I contributed significantly to several open-source software projects simplifying and accelerating the research and application of deep learning: Theano (Al-Rfou et al., 2016) as a regular contributor, Lasagne (Dieleman et al., 2015) as a lead developer and cudamat (Mnih, 2009) as a maintainer. This helps both newcomers and established researchers in their work, across different fields.

1.3 Publications

The main chapters of this thesis build on the following publications:

- J. Schlüter and R. Sonnleitner. Unsupervised feature learning for speech and music detection in radio broadcasts. In *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx)*, York, UK, Sept. 2012. URL `http://jan-schlueter.de/pubs/2012_dafx.pdf`.

- J. Schlüter. Learning binary codes for efficient large-scale music similarity search. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, Curitiba, Brazil, Nov. 2013. URL `http://jan-schlueter.de/pubs/2013_ismir.pdf`.

- J. Schlüter and S. Böck. Musical onset detection with convolutional neural networks. In *6th International Workshop on Machine Learning and Music (MML), in conjunction with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, Prague, Czech Republic, Sept. 2013. URL `http://jan-schlueter.de/pubs/2013_mml.pdf`.

- J. Schlüter and S. Böck. Improved musical onset detection with convolutional neural networks. In *Proceedings of the 39th IEEE International Con-*

ference on Acoustics, Speech, and Signal Processing (ICASSP), pages 6979–6983, Florence, Italy, May 2014. doi:10.1109/ICASSP.2014.6854953. URL http://jan-schlueter.de/pubs/2014_icassp.pdf.

- K. Ullrich, J. Schlüter, and T. Grill. Boundary detection in music structure analysis using convolutional neural networks. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, Taipei, Taiwan, Oct. 2014. URL http://jan-schlueter.de/pubs/2014_ismir.pdf.

- J. Schlüter and T. Grill. Exploring data augmentation for improved singing voice detection with neural networks. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, Málaga, Spain, Oct. 2015. URL http://jan-schlueter.de/pubs/2015_ismir.pdf.

- J. Schlüter. Learning to pinpoint singing voice from weakly labeled examples. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, New York City, NY, USA, Aug. 2016. URL http://jan-schlueter.de/pubs/2016_ismir.pdf.

For a complete list of publications published during (and before) my thesis work, please refer to the curriculum vitae at the very end of this document.

1.4 Thesis Outline

The next three chapters give an extensive didactic introduction to the methods employed in this thesis.

Chapter 2 explains the basic concepts of machine learning, the idea behind deep learning, and the nuts and bolts of applying deep learning in practice: neural network architectures, loss functions, backpropagation, and modern methods for optimization, initialization and regularization.

Chapter 3 introduces relevant elements of audio signal processing: spectrograms, perceptually-informed frequency and magnitude transformations, and higher-level audio features.

Chapter 4 connects the two chapters to discuss several alternatives for applying neural networks to audio signals.

Chapters 5 to 9 form the main part of the thesis. Each addresses a particular music perception task, and each is based on one or two of the publications listed in the previous section, with extended explanations, additional illustrations or results and an extra section describing both successful and unsuccessful follow-up work.

The chapters are presented in chronological order, so the changes in chosen methodology reflect both my learning curve and advances in the field of deep learning I tried to keep up with.

Chapter 5, based on Schlüter and Sonnleitner (2012), shows how training a mean-covariance Restricted Boltzmann Machine (mcRBM) on mel spectrograms of radio broadcast recordings yields features discriminating music and speech, without requiring any labelled data. When extended to a deeper neural network and trained on human annotations, it outperforms three existing approaches based on higher-level audio features in detecting music and speech. The features learned by the mcRBM resemble existing hand-designed features for music and speech detection, suggesting that the advantage lies in using hundreds of minor variations. Follow-up experiments with more powerful generative models remain unsuccessful.

Chapter 6, based on Schlüter (2013), employs deep auto-encoders to compress state-of-the-art music similarity models to binary codes, accelerating music similarity search to a level required to handle commercial-scale catalogues of tens of millions of items. This task is especially interesting due to the similarity measures being highly non-metric, failing a requirement of common indexing methods. Results on up to 1.1 million items outperform seven existing approaches for approximate nearest neighbour search in terms of the time required to answer a query at a given target nearest neighbour recall. Follow-up experiments demonstrate further ways to accelerate a state-of-the-art music similarity measure, and the fallibility of evaluating with genre labels.

Chapter 7, based on Schlüter and Böck (2013, 2014), tackles musical onset detection with Convolutional Neural Networks (CNNs) trained on multi-resolution mel spectrograms. Initial results slightly surpass the previous state of the art, and blurring the training targets, adding dropout and using linear rectifiers further improves results. An examination of the network shows that its solution is based on detecting temporal differences and specializing towards percussive and harmonic onsets – both in common with existing hand-designed approaches –, but using many minor variations of the same ideas, more than possible to design by hand. Follow-up work trying to include phase information, preprocessing inputs with a harmonic-percussive separation stage or adding recurrent layers did not improve results.

Chapter 8, based on Ullrich et al. (2014), extends the onset detection approach to detect musical boundaries. With an increased temporal context, reduced temporal resolution and stronger blurring of training targets compared to the onset detection network it strongly outperforms all 21 approaches submitted to an international boundary detection evaluation campaign in 2012–2014.

An examination shows that, similar to the onset detection network, the solution combines many minor variations of a handful of ideas, one of which being the detection of pauses. Follow-up work outside of this thesis further improves results by including a task-specific higher-level input feature and making use of additional forms of training data.

Chapter 9, based on Schlüter and Grill (2015) and Schlüter (2016), uses singing voice detection with CNNs trained on spectrograms as a test bed for developing two generic methods: musical data augmentation, to artificially enhance the diversity of small training sets, and a recipe for training on song-wise annotations and still produce temporally accurate predictions, or even detect which time-frequency bins contain singing voice. Both methods are experimentally verified, and although not a central goal, outperform the previous state of the art in singing voice detection. The examination of a trained CNN shows that it mostly relies on detecting diagonal or wiggly lines in a spectrogram and can easily be fooled. Follow-up work resolves this fallacy using data augmentation, experiments with stereo input and learning spectrogram and filterbank parameters, and improves results with architectural changes.

Finally, Chapter 10 concludes the thesis and suggests directions for future work on a higher level than given in the task-specific chapters.

A short appendix describes some commercial applications of my thesis work (Appendix A), and an implementation trick for efficiently using a CNN that was trained on short spectrogram excerpts to process a full recording at once (Appendix B).

2 A Primer on Deep Learning

To provide the necessary background for following the work in this thesis, this chapter explains the central ideas and techniques of contemporary deep learning. We will start by reviewing the purpose and challenges of machine learning, and then delve into the specifics of deep learning and modern artificial neural networks. Finally, I will give a timeline of important developments published before and during this thesis work, setting each chapter of this thesis in context to the respective progress of the field.

Note that this introduction is written from an engineering perspective, focusing on the mechanics required to understand and use deep learning, and deliberately omitting parts of the formal background. For a more thorough introduction to machine learning, see the text books of Bishop (2006) or Barber (2012), and for deep learning in particular, see Goodfellow et al. (2016), Karpathy et al. (2016), or Nielsen (2015).

2.1 Machine Learning

As deep learning is a branch of machine learning, we will set the stage by covering the basics of machine learning, and its two key challenges: optimization and generalization.

2.1.1 General Idea

Machine learning combines two basic ideas: (1) Formalizing a task in a way that its solution can be expressed as a mathematical function, and (2) tuning parts of this function from data.

Formalizing the task allows us to tackle it with a machine at all. For example, the task of beating Garry Kasparov in chess could be formalized as defining a mathematical function that maps a list of all chess piece positions to a move instruction. Once this function is defined, it can be implemented as a computer program[1] and executed to play against Kasparov, with some additional engineering or manual intervention to produce the program's input from a physical chess board, and execute a move according to the program's output.

As we will see, it often suffices to only consider formalizations that can be solved by a function

$$\boldsymbol{y} = f(\boldsymbol{x}), \tag{2.1}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are tensors (e.g., vectors, matrices, or just scalars) representing the input and output, respectively. This simple framework covers the following broad categories of tasks:

Regression, inferring one or more scalar values from a given data point, can be directly solved by a function mapping the data to a scalar or vector.
For example, predicting the amount of rainfall in the next ten minutes from a range of meteorological measurements can be formalized as a function computing y, the rainfall in millimetres, from $\boldsymbol{x}$, a vector of measurements.

Binary Classification, categorizing data points into one of two classes, can be solved by a function mapping the data to a number between zero and one indicating class membership.
As an example, distinguishing greyscale photographs of chihuahuas and blueberry muffins can be formalized as a function computing y, the probability of the picture showing a dog, from $\boldsymbol{X}$, a matrix of brightness value per pixel.

Categorical Classification, categorizing data points into one of K classes, can be solved by a function mapping the data to a vector of K numbers giving the respective probabilities of the data point belonging to each of the classes.

There are other ways of formalizing tasks to be solvable by a mapping from tensors to tensors, and there are other tasks we could formalize in this way, but these are common choices we will justify and keep referring to later.

[1] Not every mathematical function can be computed by a computer program, so we will want to design the function such that it can be computed, and can be computed efficiently.

Tuning the solution from data constitutes the learning aspect in "machine learning". It requires a very simple measure: When defining the function f solving a particular task, leave some parameters unspecified. These could be constants involved in an operation on the inputs, which components of the input to use, or even which operations to perform. Let us extend the definition of f to make this explicit:

$$\boldsymbol{y} = f(\boldsymbol{x}; \Theta), \tag{2.2}$$

where Θ is a tuple of parameter tensors $\boldsymbol{\theta}$. The mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$ is now dependent on the values of Θ. Requiring parameters to take the form of tensors may seem restrictive, but allows any use from defining coefficients to determining which of a given set of operations to perform (by including case distinctions in the function definition).

The hope is that once such a tunable function – or *model* – f is defined, we can automatically optimize its parameters Θ to solve a task, arriving at a better solution than found by a fully manually-specified function, or at least requiring less effort. In the following two sections, we will see what challenges need to be overcome to achieve this, and better understand the design space of machine learning algorithms. This will make clear why there is no universal model, and prepare us for the specific choices made in deep learning.

2.1.2 Optimization

In order to optimize the parameters Θ of our model f, we need to formalize a measure of how well a particular choice of model and parameters solves a given task. This formalization takes the form of a loss function

$$l = L(\Theta; f) \tag{2.3}$$

computing a scalar loss value, with lower values indicating better solutions. The formalization of the loss is closely connected to the formalization of the task.

Similarly to the model f, the loss function is not fully constructed by hand, but leaves some values to be filled in: a set of training data $\mathcal{D}$. Specifically, it often takes the form of comparing the prediction $\boldsymbol{y}$ of the model against a target value $\boldsymbol{t}$ for each input $\boldsymbol{x}$:

$$L(\Theta; f, \mathcal{D}) = \sum_{(\boldsymbol{x},\boldsymbol{t})\in\mathcal{D}} J\Big(f(\boldsymbol{x}; \Theta),\, \boldsymbol{t}\Big) \tag{2.4}$$

This simplifies the design to defining a function $J(\boldsymbol{y}, \boldsymbol{t})$ computing the penalty for a given prediction compared to a known good answer. For example, for scalars, it could be $J(y, t) = |y - t|$.

Having defined both a model and a loss, learning from data reduces to function minimization. Specifically, we want to find the parameters Θ^* minimizing the loss:

$$\Theta^* = \arg\min_{\Theta} L(\Theta; f, \mathcal{D}) \tag{2.5}$$

How this is done – and how difficult it is – depends on the combination of the chosen model and the chosen loss function. We can distinguish the following cases:

Convex: If the loss function and model form a strictly convex function of Θ, there is a single global optimum and no saddle point. In this case, the optimal solution can either be found analytically or with a convex optimization method (such as quadratic programming). For example, a linear model $f(\boldsymbol{x}; \boldsymbol{a}, b) = \boldsymbol{a}^T\boldsymbol{x} + b$ and quadratic penalty $J(y, t) = (y - t)^2$ results in a convex function L.

Nonconvex, differentiable: For more complex models or loss functions, the minimization target can become nonconvex. However, if the model f is differentiable wrt. its parameters Θ and the penalty J is differentiable wrt. the prediction $\boldsymbol{y}$, then L is also differentiable wrt. Θ (using the chain rule). In this case, we can find a local optimum $\hat{\Theta}$ via a gradient-based method such as gradient descent.

Nonconvex, nondifferentiable: The most general case is a nonconvex target function that is not differentiable with respect to the model parameters. This requires stochastic search methods such as evolutionary algorithms.

From the first to the last, the cases increase in difficulty, typically requiring more computational effort to find a solution and at the same time loosing guarantees on the optimality of the solution being found.

A major challenge in machine learning lies in designing the model powerful enough to produce good predictions, but in a way that the tandem of model and penalty function are simple enough for efficient optimization. If the model is too simple or if it is difficult to optimize, it may not be able to produce good predictions on the training data – this failure case is called *underfitting*.

Figure 2.1 demonstrates this for a toy problem. Given a dataset of 20 scalar inputs with scalar targets (shown as blue dots), we want to learn a mapping from inputs to targets that matches the training data. We define our model to be a third order polynomial $f(x; \boldsymbol{a}) = \sum_{i=0}^{3} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ with learned coefficients $\boldsymbol{a}$, and the penalty to be the squared difference between prediction and target $J(y, t) = (y - t)^2$. The optimal solution can be found analytically. However, plotting the predictions of the optimal solution for all inputs $x \in [0, 5]$ (shown as a red line), we see that they do not match the training examples too well: The model is too simple and underfits the data.

Finding a powerful enough optimizable model is only part of the challenge, though, as we will see in the next section.

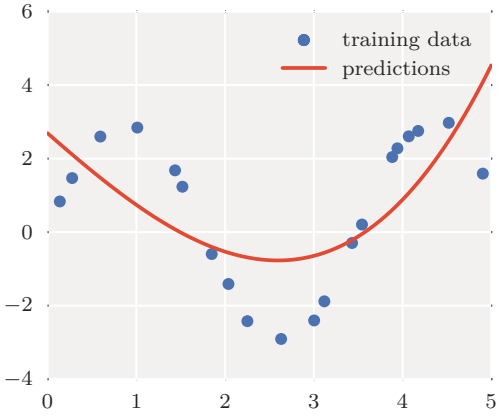


Figure 2.1: A too simple model may *underfit* the training data, as explained in Sec. 2.1.2.

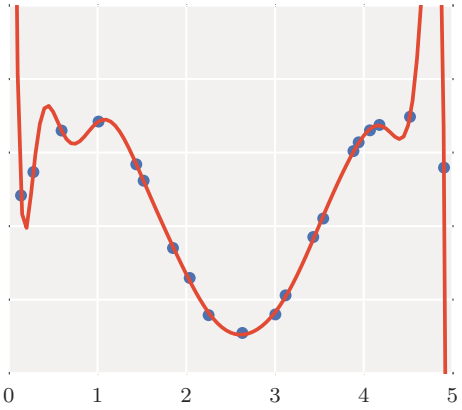


Figure 2.2: A too complex model may *overfit* the training data, as explained in Sec. 2.1.3.

2.1.3 Generalization

For difficult tasks, it is not sufficient to merely find a model that gives good predictions on a set of training examples – this could be easily achieved by a look-up table, and would not require machine learning at all. Instead, we need a model that even gives good predictions *outside* the set of training examples. For example, to beat Kasparov in chess, a model needs to suggest a good move for any board position occurring in a match against Kasparov, which will most likely include positions not used in training the model – otherwise the task would be easy.

This is what distinguishes machine learning from plain function minimization as was discussed in the previous section: The challenge is to design the model, loss function and optimization method in a way that the solution *generalizes* to unseen data. It is well possible that after training, a model performs well on the training data only – this failure case is called *overfitting*.

Figure 2.2 demonstrates this for our toy problem. To produce better predictions than the third-order polynomial model of Figure 2.1, we replace the model by a 15th-order polynomial $f(x; \boldsymbol{a}) = \sum_{i=0}^{15} a_i x^i$, optimize it on our 20 training examples (blue dots), and compute the predictions for all inputs $x \in [0, 5]$ (red line). It achieves near-zero loss on the training data, but meanders suspiciously near the ends of the input range. Of course, plotting predictions over the input space is not always feasible, and suspicion is subjective. To formally check for overfitting, we need to compare predictions to ground truth for unseen examples. Assuming we cannot obtain additional examples, we designate three of the training examples as our *validation set* and re-train the model on the remaining 17 examples.

Figure 2.3: Sign for overfitting: Predictions match training examples much better than unseen validation examples.

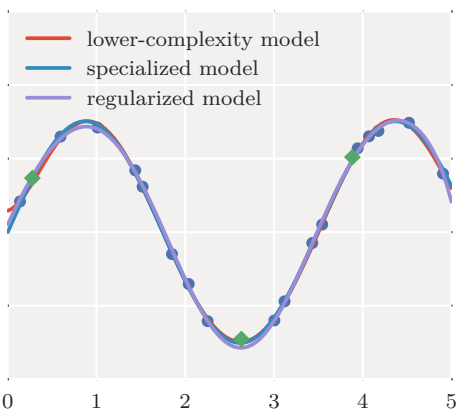


Figure 2.4: Overfitting can be reduced by lowering model complexity, choosing a specialized model, or regularization.

Figure 2.3 shows the result: The solution – slightly different due to the changed training set – achieves near-zero loss on the training examples, but performs poorly on some of the unseen validation examples (which can be quantified by the loss function $L(\Theta; f, \mathcal{D})$, setting $\mathcal{D}$ to the validation set). The model has overfit.

There are several ways to combat overfitting, and conversely, enhance generalization. The first three are demonstrated in Figure 2.4, with almost identical results.

Lowering model complexity: Reducing the number of free parameters of a model makes it harder to learn examples by heart, and thus encourages solutions that exploit some regularities in the training data in order to approximate it well. If lucky, these regularities carry over to unseen data.
For our toy example, reducing the model to a 6th-order polynomial avoids overfitting, indicated by good predictions on the validation set.

Choosing a specialized model: Sometimes we know enough about a task to have an idea of what a solution should look like, and can use this to restrict the search space. This could involve preprocessing the inputs or postprocessing the outputs, simplifying what needs to be learned by the model, or designing the model itself to be specialized for the task at hands.
In our toy example, we may guess that the data could be modelled by a sinusoid function.[2] Setting $f(x; a, b) = a \sin(bx)$ and optimizing (with a gradient-based method), we indeed obtain a good solution, despite using even fewer parameters than the third-order polynomial of Figure 2.1.

[2] This would be a good guess; the training data was generated as $t = 3 \sin(1.8x) + 0.1 \sin(10x)$.

Regularization: Starting from very general assumptions about how solutions should look like, one can derive so-called regularization methods that are largely independent of the task and model.
A common assumption is that a mapping should change slowly over the input space, i.e., have a small gradient everywhere, and no sharp changes or oscillations, capturing the intuition that similar inputs should lead to similar predictions. This can be accomplished by choosing small values for any parameters multiplied with an input value: This limits the effect a small change of the input can have on the output. In practice, this can be expressed as an additional penalty term R in the loss function:

$$L(\Theta; f, \mathcal{D}) = \sum_{(\boldsymbol{x},\boldsymbol{t})\in\mathcal{D}} J(f(\boldsymbol{x};\Theta),\boldsymbol{t}) + R(\Theta) \tag{2.6}$$

If we set $R(\Theta)$ to be the sum of squares of all parameters,

$$R(\Theta) = \lambda \sum_{\boldsymbol{\theta}\in\Theta} \|\boldsymbol{\theta}\|_F^2, \tag{2.7}$$

scaled by some factor λ, the optimal solution Θ^* will be driven to use small parameter values. λ balances the parameter penalty against the prediction penalties and needs to be chosen as part of the loss function definition – if set too low, it will not have any effect, if set too high, it can cause underfitting. This form of regularization is called L_2 regularization, as it penalizes the squared L_2 norm of parameter vectors. Other forms of regularization directly constrain parameters to a maximum value, or try to limit the number of input features used by the solution (assuming that simpler mappings involving fewer inputs will be more general).
In our toy example, setting $R(\boldsymbol{a}) = 0.0002 \sum_{i=0}^{15} a_i^2$ for the 15th-order polynomial model results in a solution with smaller coefficients that avoids large oscillations and noticeably improves predictions on the validation set.

Adding more training data: Another very general way to improve generalization is to increase the amount of training examples. This adds more constraints on the parameters of the solution, ruling out mappings that only explain the previous, smaller training set, and thus making it more likely to arrive at a universally useful solution. It can also be seen as the inverse measure to reducing the complexity of the model – both aim to decrease the ratio of free parameters to parameter constraints.

Having covered the basic ideas behind and the challenges of machine learning, we will now continue to the specific field of deep learning, and review its approaches to these challenges.

2.2 Deep Learning and Neural Networks

Deep learning is a subfield of machine learning. As such, it commits to a particular set of design choices, and explores ways of addressing the two challenges of machine learning in this self-imposed design space. In doing so, it has accumulated a set of methods that have proven to work well across a large range of tasks, sometimes outperforming other approaches by a large margin. In the following, we will review the design choices and their implications, as well as the most important methods in contemporary deep learning.

2.2.1 General Idea

The central design choice in deep learning is to use a tunable function $f(\boldsymbol{x})$ that consists of multiple stacked nonlinear operations. Such a tunable function is called a *deep* model, and its depth is the number of stacked nonlinearities.

Using nonlinear operations is what makes stacking meaningful at all: Stacking two linear functions $a(\boldsymbol{x})$ and $b(\boldsymbol{a})$ results in a function $c(\boldsymbol{x}) = b(a(\boldsymbol{x}))$ that is itself linear, and can be expressed as a single linear operation.

Stacking multiple operations means the model is composed of simpler functions that form a processing chain. This opens up the possibility for the computation to proceed from input to prediction in incremental abstractions, and to reuse intermediate results. As a formally proven advantage, this reuse allows some mappings to be expressed exponentially more compactly as a deep function than as a shallow one (Hastad, 1986; Bengio and Delalleau, 2011). Moreover, both abstractions and reuse are common features of human-engineered solutions to complex tasks, and it is appealing to use a model that could replicate aspects of human solutions, even if there is no guarantee that it does.

A central design pattern in deep learning is to try to replace hand-engineered parts of a system by a deep enough model, and have it learn a solution that improves over the engineered one. Taken to the extreme, this results in *end-to-end learning*: Having a single deep model learn a mapping from raw inputs to predictions. Since this completely avoids feature engineering, it can even be applied in cases where there is no known hand-engineered solution in the first place. Of course this design pattern is a tradeoff: While it potentially requires less expert knowledge in the task domain and less manual engineering, it requires more expertise in deep learning, more experimentation and more careful curation of data.

Without further constraints, the design space for constructing a deep tunable function is huge. Modern deep learning has converged on a common set of nonlinear operations that have proven convenient for building and training deep models. Functions built from these operations are termed artificial neural networks. In the remainder of this chapter, we will investigate this type of model in detail.

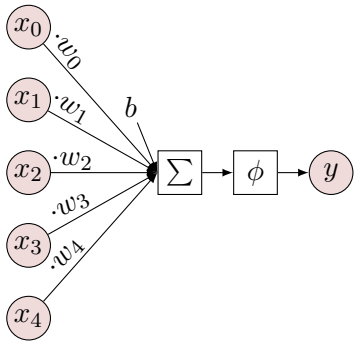


Figure 2.5: Visualization of $\phi\left(b + \boldsymbol{w}^T\boldsymbol{x}\right)$.

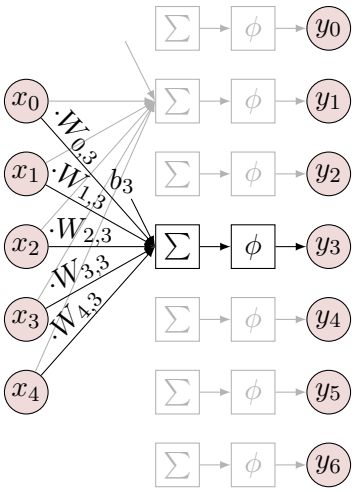


Figure 2.6: Visualization of $\phi\left(\boldsymbol{b} + \boldsymbol{W}^T\boldsymbol{x}\right)$.

2.2.2 Multi-Layer Perceptron (MLP)

About the simplest function of the family of artificial neural networks is

$$d(\boldsymbol{x}; \Theta, \phi) = \phi\Big(b + \sum_i w_i x_i\Big) = \phi\left(b + \boldsymbol{w}^T\boldsymbol{x}\right), \tag{2.8}$$

where $\phi(a)$ is some predefined function, and $\Theta = (b, \boldsymbol{w})$ are the tunable parameters: A weight vector $\boldsymbol{w}$, and a bias term b. It maps an input vector $\boldsymbol{x}$ to a scalar y by computing a weighted sum of the input values x_i, expressed as a dot product $\boldsymbol{w}^T\boldsymbol{x}$, adding a scalar offset b and passing it through the (often nonlinear) function $\phi(a)$.

In the terminology of artificial neural networks, this operation is referred to as a neuron, or unit. When visualized as a graph (Figure 2.5), it becomes apparent why: It has incoming connections from a set of nodes which can be likened to dendrites and synapses, it accumulates all incoming excitatory and inhibitory signals, and fires with a strength that depends nonlinearly on the accumulated input. Of course the similarity to biological neurons is highly superficial, and it is more instructive to treat the function as a dot product followed by a nonlinearity. However, we will borrow the terminology of units and connections when convenient.

To obtain a mapping from vectors $\boldsymbol{x}$ to vectors $\boldsymbol{y}$, we simply use multiple units of the same form, each with a separate bias and set of weights (Figure 2.6). Since all these units share the same inputs $\boldsymbol{x}$, we can express the vector of weighted sums as a matrix product $\boldsymbol{W}^T\boldsymbol{x}$. Using a vector addition for the biases, we arrive at

$$D(\boldsymbol{x}; \Theta, \phi) = \phi\left(\boldsymbol{b} + \boldsymbol{W}^T\boldsymbol{x}\right). \tag{2.9}$$

With a suitable choice of the so-called transfer function $\phi(\cdot)$, this can express solutions for all three categories of tasks mentioned in Section 2.1.1 (p. 8):

Regression: With $\phi(a) := a$, Equation 2.8 maps the input to a real-valued scalar.

Binary Classification: When setting $\phi(a)$ to the logistic sigmoid function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}, \tag{2.10}$$

then Equation 2.8 maps the input to a value between zero and one that can be interpreted as the probability of the input belonging to the first class. This corresponds to the model $p(y = 1|\boldsymbol{x})$ for logistic regression of Cox (1958). Alternatively, with $\phi(a) := \operatorname{sgn}(a)$, it is equivalent to the learnable part of the *Perceptron* proposed by Rosenblatt (1958), mapping the input to -1 or $+1$.

Categorical Classification: Using as many output units as there are classes and choosing $\phi(\boldsymbol{a})$ to be the softmax function

$$s(\boldsymbol{a})_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)}, \tag{2.11}$$

our Equation 2.9 maps input vectors to a vector of class probabilities that sum to one.

Now this model still has a serious limitation: It can only learn linear relations. It performs a linear projection of the input vector $\boldsymbol{x}$ onto the hyperplane(s) defined by $\boldsymbol{w}$ or $\boldsymbol{W}$ and directly returns the result for regression, or squashes or thresholds it for classification. However, we obtain a nonlinear model simply by stacking it:

$$D\left(D\left(\boldsymbol{x}; \Theta_1, \phi_1\right); \Theta_2, \phi_2\right) = \phi_2\left(\boldsymbol{b_2} + \boldsymbol{W_2}^T \phi_1\left(\boldsymbol{b_1} + \boldsymbol{W_1}^T \boldsymbol{x}\right)\right) \tag{2.12}$$

The resulting model is called a Multi-Layer Perceptron (MLP), and each of its constituent functions D is called a layer. Again, this terminology becomes clearer when considering the graphical representation (Figure 2.7): The units are not fully interconnected, but form three groups that are connected in sequence. More specifically, the input vector $\boldsymbol{x}$ is referred to as the input layer, the outermost function as the output layer, and any functions in between are called hidden layers.

For a particular problem, the size of the output layer and its transfer function ϕ are usually determined by the task. However, the number of hidden layers can be increased at will (by stacking more functions D), their individual sizes can be freely chosen (by adapting the sizes of the weight matrices and bias vectors), and their transfer functions set to any nonlinear function. Typically, a single function is selected for all hidden units of a model. Widespread transfer functions include the sigmoid $\sigma(a)$ (Equation 2.10), the more modern rectifier (Glorot et al., 2011)

$$r(a) = \max(a, 0) \tag{2.13}$$

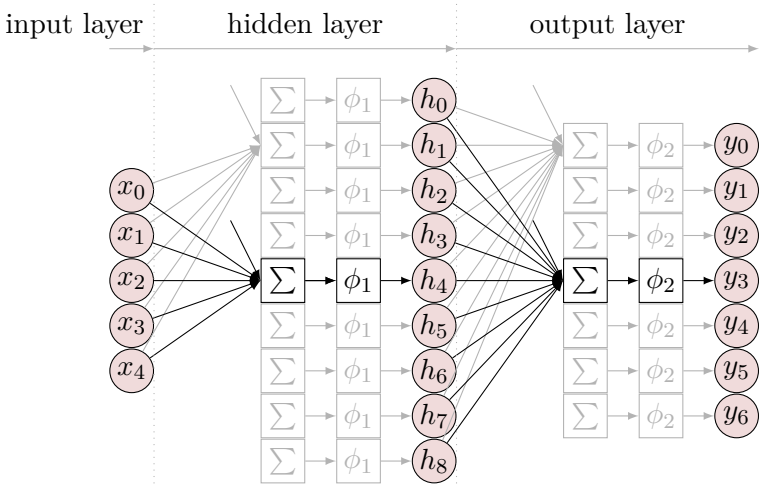
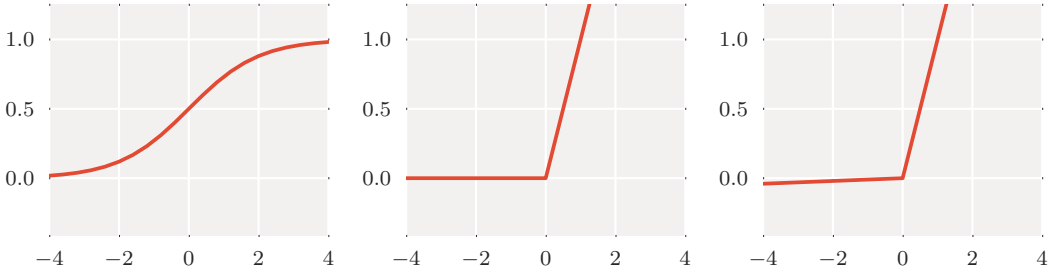


Figure 2.7: Visualization of $\phi_2\left(\boldsymbol{b_2} + \boldsymbol{W_2}^T \phi_1\left(\boldsymbol{b_1} + \boldsymbol{W_1}^T \boldsymbol{x}\right)\right)$.



(a) $\sigma(a) = (1 + \exp(-a))^{-1}$ (b) $r(a) = \max(a, 0)$ (c) $l(a) = \max(a, 0.01a)$

Figure 2.8: Typical nonlinear transfer functions for hidden layers.

and the leaky rectifier (Maas et al., 2013)

$$l(a) = \max\left(a, \frac{a}{100}\right), \tag{2.14}$$

shown in Figure 2.8.

In theory, even an MLP of a single, large enough hidden layer with a sigmoid transfer function – i.e., precisely the form given in Equation 2.12 and Figure 2.7, with $\phi_1(a) := \sigma(a)$ – can approximate any continuous function of finite support arbitrarily well (Hornik et al., 1989). In practice, it may be impossible to train, and it pays off to explore the full design space of deeper models and alternative nonlinearities. Besides, even the MLP's layer function D (Equation 2.9) can be a suboptimal choice, as we will see and address in the next section.

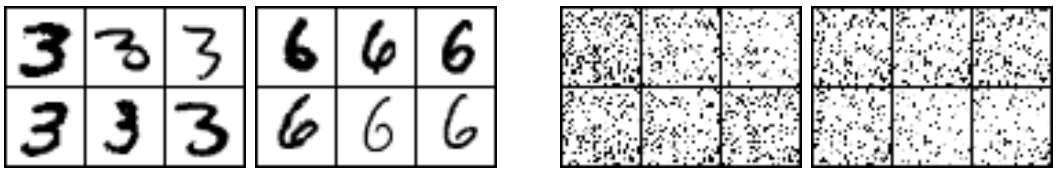


Figure 2.9: *Left:* Images of hand-written digits. *Right:* The same images with a fixed permutation of pixels. For an MLP, the pixel order on the left is as arbitrary as the right one, but for humans, it is a lot easier.

2.2.3 Convolutional Neural Network (CNN)

The layer function of the MLP discussed in the previous section, $D(\boldsymbol{x}; \Theta, \phi) = \phi(\boldsymbol{b} + \boldsymbol{W}^T \boldsymbol{x})$, has a special property: It is oblivious to the ordering of features in a dataset $\boldsymbol{x}, \boldsymbol{t} \in \mathcal{D}$. When we swap the same two components k, l of each input vector $\boldsymbol{x}$ to obtain a new dataset

$$\mathcal{D}' = \left\{ (\boldsymbol{x}', \boldsymbol{t}) \middle| (\boldsymbol{x}, \boldsymbol{t}) \in \mathcal{D} \wedge x'_k = x_l \wedge x'_l = x_k \wedge \forall_{i \notin \{k,l\}} x'_i = x_i \right\}, \tag{2.15}$$

we can swap the corresponding rows k, l of $\boldsymbol{W}$ to obtain a new weight matrix

$$W'_{i,j} = \begin{cases} W_{k,j}, & \text{if } i = l \\ W_{l,j}, & \text{if } i = k \\ W_{i,j}, & \text{otherwise} \end{cases} \tag{2.16}$$

such that the layer outputs remain the same

$$D(\boldsymbol{x}; (\boldsymbol{W}, \boldsymbol{b}), \phi) = D(\boldsymbol{x}'; (\boldsymbol{W}', \boldsymbol{b}), \phi) \tag{2.17}$$

for corresponding data points from $\mathcal{D}$ and $\mathcal{D}'$. Moreover, for most optimization methods, learning $\boldsymbol{W}$ from $\mathcal{D}$ is exactly equivalent to learning $\boldsymbol{W}'$ from $\mathcal{D}'$. The same argument can be made for the target vectors: Swapping two target vector components can be countered by swapping two columns of the weight matrix and two entries of the bias vector, if the nonlinearity ϕ is oblivious to the input ordering. By extension, we can completely scramble the order of input and target components in a dataset without making it any more difficult for an MLP to learn or express a particular solution.

This property is shared with several other machine learning models, and often desirable: If the input can be described as a set of independent numerical attributes, we do not want the presentation order to make any difference. However, for some tasks, the input features have an inherent structure, such as a temporal sequence or a 2D lattice of image pixels, that may be useful to exploit. Figure 2.9 gives a drastic example: When permuting the pixels of images of hand-written digits,

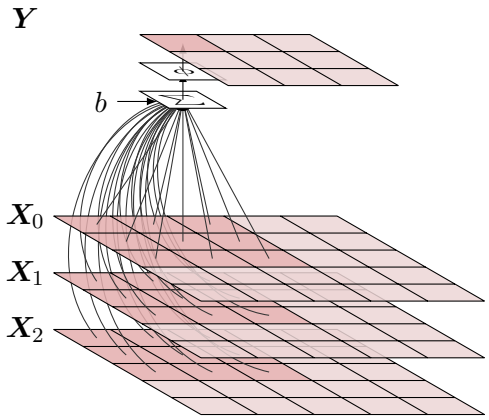


Figure 2.10: A convolutional unit.

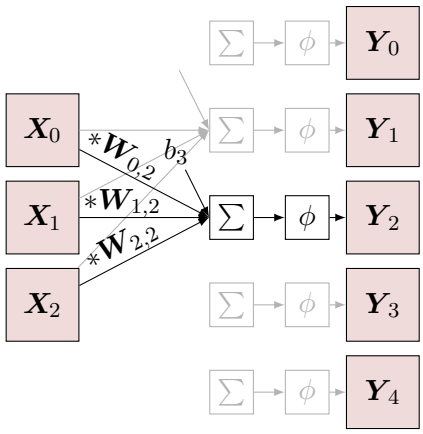


Figure 2.11: A convolutional layer.

their structure is destroyed, and they become virtually impossible to recognize for humans. For an MLP, however, applying the same permutation of pixels to each image of a dataset does not change the problem at all. Regardless of the pixel order, it has to learn from scratch which groups of pixels usually light up together, form a more abstract description of the input in terms of such groups, and finally associate different patterns with different targets.

If we know the input data consists of images, we can specialize the model to exploit their structure, simplifying the learning task. A particularly successful way is to replace the neurons in the initial layers of an MLP with *convolutional units*

$$c(\boldsymbol{X}; \Theta, \phi) = \phi(b + \sum_i \boldsymbol{X}_i * \boldsymbol{W}_i). \tag{2.18}$$

They can be derived from their MLP counterpart (Equation 2.8) by replacing each scalar input x_i with a matrix $\boldsymbol{X}_i$, each scalar weight w_i with a matrix $\boldsymbol{W}_i$, and scalar multiplication by two-dimensional convolution. Figure 2.10 visualizes this operation: The top left element of the output matrix is computed as the weighted sum of the top left regions of all input matrices, followed by a bias and nonlinearity. By repeating this for all regions of that size, each time using the same set of weights, we obtain the full output matrix. This is equivalent to convolving each input matrix $\boldsymbol{X}_i$ with a matrix $\boldsymbol{W}_i$ and adding up the results, followed by bias and nonlinearity.

Similar to how multiple neurons d form an MLP layer D, multiple convolutional units operating on the same input form a *convolutional layer* (Figure 2.11):

$$C(\boldsymbol{X}; \Theta; \phi)_j = \phi(\boldsymbol{b} + \sum_i \boldsymbol{X}_i * \boldsymbol{W}_{i,j}) \tag{2.19}$$

The input and output 3-tensors $\boldsymbol{X}$ and $\boldsymbol{Y}$ are referred to as a set of *image channels* or *feature maps*, and the weight 4-tensor $\boldsymbol{W}$ as a set of *filters* or *kernels*. Again, the

convolutional layer can be derived from the MLP layer by adding two dimensions to the input vector, output vector and weight matrix and replacing multiplication by two-dimensional convolution. The bias $\boldsymbol{b}$ remains a vector, as it is shared over all locations within a feature map.

When applied to images, the input matrices $\boldsymbol{X}_i$ represent the different colour channels (e.g., the red, green, blue intensities, or just grey values), and the output matrices represent differently processed versions of the input. While convolutions are not the only way to make the model sensitive to the order of input pixels – any operation or sparse connectivity that prevents adapting the model parameters to swapped pixels would do – they have several particularly useful properties:

Local connectivity: Each output pixel depends on a set of neighbouring input pixels. This restricts a convolutional layer to learn local features, and makes it easier to exploit the strong correlations between neighbouring pixels commonly found in images. It also strongly reduces the number of parameters compared to an MLP layer, reducing the risk of overfitting.

Weight sharing: Weights are shared across all spatial locations. This exploits the assumption that local image statistics are stationary, and further reduces the number of parameters. It also acts as a strong regularizer: Filters have to be useful across the input and cannot lock to a feature only present at a particular location.

Spatial layout: The output of a convolutional layer reflects the spatial layout of the input. Thus, convolutional layers can be stacked.

The design space for Convolutional Neural Networks (CNNs, or ConvNets) is similar to that of MLPs: We can freely choose the number of hidden layers and their individual sizes and transfer functions. However, we do not only have to fix the number of convolutional units for a layer, but also the size of the filters – that is, the rectangular shape of the region of input pixels participating in the computation of an output pixel. In Figure 2.10, the filter size is 3×3, resulting in a 3×3 output for a 5×5 input. Increasing the filter size increases the amount of spatial context per output pixel, decreases the size of the output, and increases the number of learnable parameters. In the extreme case, when setting the filter size equal to the input size, the convolutional filters can only be applied at a single position, and the layer becomes equivalent to an MLP layer: Each convolutional unit computes a single output pixel that depends on all input pixels at once. Due to this connection pattern, such a layer is referred to as a *fully-connected layer*, or *dense layer*. For most tasks, a CNN will end in one or more fully-connected layers that integrate information across all spatial locations and finally produce the prediction. This way the model is still oblivious to the ordering of target vector components, like an MLP, which is appropriate for most classification and regression tasks.

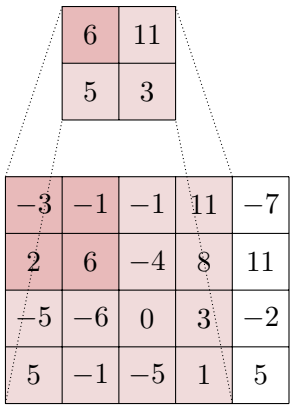


Figure 2.12: Max-pooling.

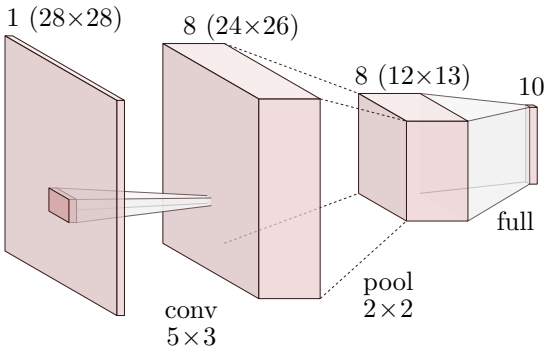


Figure 2.13: A CNN example architecture with three different layer types.

The amount of data processed by hidden layers in a CNN can quickly grow: While the first convolutional layer often processes an input of three channels or less, the next layer has to process as many channels as the first layer has convolutional units, of almost the same spatial size if filters are small. Moreover, the *receptive field* – the number of network input pixels a layer's output pixels depend on – only grows slowly throughout the network: Two 3×3 convolutions depend on 5×5 input pixels, k such convolutions have a receptive field of $(1+2k)\times(1+2k)$. Increasing the kernel size reduces both the spatial size and increases the receptive field more quickly, but also strongly increases the number of parameters: A 5×5 convolution has 178% more, a 7×7 convolution 444% more parameters than a 3×3 convolution.

As a way out, it is common to place parameter-free *subsampling* or *pooling layers* between some of the convolutional layers. For example, a 2×2 max-pooling layer only retains the maximum values of non-overlapping 2×2 input patches (see Figure 2.12), separately for each input channel. This shrinks the feature maps and enlarges the receptive field by a factor of 2 in each dimension, without adding any parameters. Alternatives include mean-pooling (retaining the average instead of the maximum per patch), or discarding every second row and column (implementable as part of the previous convolution by only applying the filters at every second input position, termed *strided convolution*). As demonstrated in Figure 2.12, if the input size is not evenly divisible by the pooling factors, part of the input will be ignored – this situation can be avoided by tuning the filter sizes of preceding convolutions.

CNN architectures thus commonly consist of three types of layers: convolutional layers, pooling layers, and fully-connected layers. Figure 2.13 shows a schematic architecture of a single layer of each of these types. It depicts feature maps as 3D volumes, with the number of channels forming the third dimension, and dense layers as columns. Labels denote the number and size of feature maps (top) and layer types (bottom). I will use this kind of visualization throughout the thesis.

2.2.4 Optimization

In the previous two sections, we have seen what the model functions of deep neural networks look like. To optimize their parameters for a task, we need both a loss function and a suitable optimization method, as discussed in Section 2.1.2 (p. 9). In the following, we will detail what these ingredients look like, what challenges are posed by the depth of the models, and how these are overcome.

2.2.4.1 Loss Functions

Recalling Equation 2.4 (p. 9), $L(\Theta; f, \mathcal{D}) = \sum_{(\boldsymbol{x},\boldsymbol{t})\in\mathcal{D}} J\big(f(\boldsymbol{x};\Theta),\, \boldsymbol{t}\big)$, defining a loss function boils down to defining a penalty $J(\boldsymbol{y}, \boldsymbol{t})$ comparing a network prediction against a target. For the three categories of tasks introduced in Section 2.1.1 (p. 8), the following penalties are commonly used together with the network output nonlinearities given in Section 2.2.2 (p. 16):

Regression uses the squared error,

$$J(y, t) := (y - t)^2. \tag{2.20}$$

Binary classification uses binary cross-entropy,

$$J(y, t) := -t \log(y) - (1 - t) \log(1 - y). \tag{2.21}$$

Categorical classification uses categorical cross-entropy,

$$J(\boldsymbol{y}, \boldsymbol{t}) := -\sum_i t_i \log(y_i). \tag{2.22}$$

These choices are not arbitrary, but can be justified from a probabilistic interpretation. This assumes that our training examples $(\boldsymbol{x}, \boldsymbol{t}) \in \mathcal{D}$ are independent random samples from an unknown distribution $P(\boldsymbol{x}, \boldsymbol{t})$, and our objective is to find a model $Q(\boldsymbol{t}|\boldsymbol{x}; \Theta)$ of the conditional distribution $P(\boldsymbol{t}|\boldsymbol{x})$, to predict $\boldsymbol{t}$ from $\boldsymbol{x}$. Specifically, we want to maximize the likelihood of the training examples under our model, which is equivalent to minimizing the negative log likelihood:

$$\Theta^* = \arg\max_\Theta \prod_{(\boldsymbol{x},\boldsymbol{t})\in\mathcal{D}} Q(\boldsymbol{t}|\boldsymbol{x}; \Theta) \tag{2.23}$$

$$= \arg\min_\Theta \sum_{(\boldsymbol{x},\boldsymbol{t})\in\mathcal{D}} -\log Q(\boldsymbol{t}|\boldsymbol{x}; \Theta) \tag{2.24}$$

This already gives us the form of Equation 2.4, with $\mathrm{J}\big(f(\boldsymbol{x};\Theta),\, \boldsymbol{t}\big) = -\log Q(\boldsymbol{t}|\boldsymbol{x}; \Theta)$. Different choices of the model Q directly lead to the different penalty functions. For regression, if we define $Q(t|\boldsymbol{x}; \Theta)$ to be a Gaussian distribution $\mathcal{N}\left(t|f(\boldsymbol{x};\Theta), \sigma^2\right)$

with mean $f(\boldsymbol{x};\Theta)$ given by the neural network and fixed variance $\sigma^2 = 0.5$, the negative log likelihood equals the squared error plus some terms not relevant for optimizing Θ (Bishop, 2006, Eq. 1.62). For binary classification, defining $Q(t|\boldsymbol{x};\Theta)$ to be a Bernoulli distribution with success probability $Q(t=1|\boldsymbol{x};\Theta) = f(\boldsymbol{x};\Theta)$ given by the neural network and taking the negative log likelihood recovers the binary cross-entropy (Bishop, 2006, Eq. 2.6). Finally, defining $Q(\boldsymbol{t}|\boldsymbol{x};\Theta)$ to be a categorical distribution with event probabilities $Q(t_i=1|\boldsymbol{x};\Theta) = f(\boldsymbol{x};\Theta)_i$ given by the neural network and again computing the negative log likelihood yields the categorical cross-entropy (Bishop, 2006, easy to see from Eq. 2.29).

2.2.4.2 Backpropagation

As established in Section 2.1.2, the tandem of model function and loss function involved in optimizing the model parameters Θ can fall into either of three categories: (1) convex, (2) nonconvex and differentiable, or (3) nonconvex and nondifferentiable. Due to the stacked nonlinearities in a deep model, $L(\Theta; f, \mathcal{D})$ is not a convex function of Θ, ruling out the first category that is easiest to optimize. However, all penalty functions $J(\boldsymbol{y}, \boldsymbol{t})$ of the previous section are differentiable wrt. $\boldsymbol{y}$, and in the design of neural networks, great care has been taken that their model functions are differentiable wrt. Θ. Taken together, this allows to differentiate the loss function wrt. the model parameters:

$$\frac{\partial}{\partial\Theta}L(\Theta; f, \mathcal{D}) = \sum_{(\boldsymbol{x},\boldsymbol{t})\in\mathcal{D}} \frac{\partial}{\partial\Theta}J\Big(f(\boldsymbol{x};\Theta),\, \boldsymbol{t}\Big) = \sum_{(\boldsymbol{x},\boldsymbol{t})\in\mathcal{D}} \frac{\partial}{\partial\boldsymbol{y}}J(\boldsymbol{y},\boldsymbol{t})\frac{\partial}{\partial\Theta}f(\boldsymbol{x};\Theta) \tag{2.25}$$

Here, $\partial L/\partial\Theta$ denotes a row vector of all partial derivatives of the scalar loss wrt. the different model parameters (a gradient). $\partial J/\partial\boldsymbol{y}$ is the gradient wrt. the network outputs. $\partial f/\partial\Theta$ denotes a matrix of all partial derivatives of the different network outputs wrt. the different model parameters (a Jacobian matrix). Multiplied, they yield the gradient $\partial J/\partial\Theta$. We will now have a closer look at how to compute these.

For reasons that will become clear shortly, let us separate the network's output transfer function $\boldsymbol{y} = \phi(\boldsymbol{a})$ from the remaining network $\boldsymbol{a} = g(\boldsymbol{x};\Theta)$, such that $f(\boldsymbol{x};\Theta) = \phi(g(\boldsymbol{x};\Theta))$. Using this, we can split up the gradient differently:

$$\frac{\partial}{\partial\boldsymbol{y}}J(\boldsymbol{y},\boldsymbol{t})\frac{\partial}{\partial\Theta}f(\boldsymbol{x};\Theta) = \frac{\partial}{\partial\boldsymbol{y}}J(\boldsymbol{y},\boldsymbol{t})\frac{\partial}{\partial\boldsymbol{a}}\phi(\boldsymbol{a})\frac{\partial}{\partial\Theta}g(\boldsymbol{x};\Theta) = \frac{\partial}{\partial\boldsymbol{a}}J\Big(\phi(\boldsymbol{a}),\, \boldsymbol{t}\Big)\frac{\partial}{\partial\Theta}g(\boldsymbol{x};\Theta) \tag{2.26}$$

As it turns out, for usual combinations of output transfer function ϕ and penalty J (Sections 2.2.2 and 2.2.4.1, respectively), $\frac{\partial}{\partial\boldsymbol{a}}J(\phi(\boldsymbol{a}),\, \boldsymbol{t})$ has a very simple form: For regression with linear output and squared error, it is $2(y-t)$, for binary classification with sigmoid output and binary cross-entropy, it is $(y-t)$, and for categorical classification with softmax output and categorical cross-entropy, it is $(\boldsymbol{y}-\boldsymbol{t})^T$. For the derivations, refer to Bishop (2006, Sec. 4.3.6), for example.

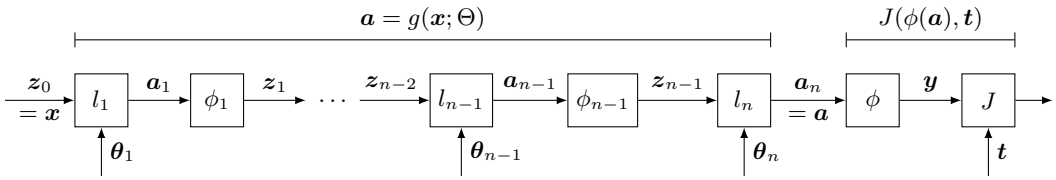


Figure 2.14: Computational graph of the minimization target $J(\boldsymbol{y}, \boldsymbol{t})$, split into $J(\phi(\boldsymbol{a}), \boldsymbol{t})$ and the remaining deep network $\boldsymbol{a} = g(\boldsymbol{x}; \Theta)$.

To compute the Jacobian of the remaining network function $\boldsymbol{a} = g(\boldsymbol{x}; \Theta)$ wrt. Θ, we can again make use of the chain rule. As we have seen in Sections 2.2.1 to 2.2.3, a deep neural network is a stack of nonlinear layers, which in turn consist of a parameterized linear transformation $l(\boldsymbol{z}; \boldsymbol{\theta})$ and a transfer function $\phi(\boldsymbol{a})$:

$$g(\boldsymbol{x}; \Theta) = l_n(\phi_{n-1}(l_{n-1}(\ldots(\phi_1(l_1(\boldsymbol{x}; \boldsymbol{\theta}_1))\ldots); \boldsymbol{\theta}_{n-1})); \boldsymbol{\theta}_n) \tag{2.27}$$

$$= (l_n \circ \phi_{n-1} \circ l_{n-1} \circ \ldots \circ \phi_1 \circ l_1)(\boldsymbol{x}; \boldsymbol{\theta}_n, \boldsymbol{\theta}_{n-1}, \ldots, \boldsymbol{\theta}_1) \tag{2.28}$$

To ease notation, let $\boldsymbol{z}_0 = \boldsymbol{x}$ denote the input to the first layer, let $\boldsymbol{a}_k = l_k(\boldsymbol{z}_{k-1}; \boldsymbol{\theta}_k)$ denote the activation of layer k, and let $\boldsymbol{z}_k = \phi_k(\boldsymbol{a}_k)$ denote the output of layer k. See Figure 2.14 for a visualization of this computation. Let us now first consider the Jacobian wrt. the parameters $\boldsymbol{\theta}_n$ of the last layer:

$$\frac{\partial}{\partial \boldsymbol{\theta}_n} g(\boldsymbol{x}; \Theta) = \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{\theta}_n} = \frac{\partial}{\partial \boldsymbol{\theta}_n} l_n(\boldsymbol{z}_{n-1}; \boldsymbol{\theta}_n) \tag{2.29}$$

Since $\boldsymbol{z}_{n-1}$ is not a function of $\boldsymbol{\theta}_n$, this means whatever this Jacobian looks like (we will come to this later), it can be computed just from the input of the layer. Proceeding to the second-to-last layer, we have:

$$\frac{\partial}{\partial \boldsymbol{\theta}_{n-1}} g(\boldsymbol{x}; \Theta) = \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{\theta}_{n-1}} = \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{a}_{n-1}} \frac{\partial \boldsymbol{a}_{n-1}}{\partial \boldsymbol{\theta}_{n-1}} = \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{z}_{n-1}} \frac{\partial \boldsymbol{z}_{n-1}}{\partial \boldsymbol{a}_{n-1}} \frac{\partial \boldsymbol{a}_{n-1}}{\partial \boldsymbol{\theta}_{n-1}} \tag{2.30}$$

$$= \frac{\partial l_n(\boldsymbol{z}_{n-1}; \boldsymbol{\theta}_n)}{\partial \boldsymbol{z}_{n-1}} \frac{\partial \phi_{n-1}(\boldsymbol{a}_{n-1})}{\partial \boldsymbol{a}_{n-1}} \frac{\partial l_{n-1}(\boldsymbol{z}_{n-2}; \boldsymbol{\theta}_{n-1})}{\partial \boldsymbol{\theta}_{n-1}} \tag{2.31}$$

So this requires the input $\boldsymbol{z}_{n-2}$ to l_{n-1}, the Jacobian of the layer's transfer function ϕ_{n-1} wrt. its input and the Jacobian of the last layer's linear transformation l_n wrt. its input. Finally, for the first layer, we get:

$$\frac{\partial}{\partial \boldsymbol{\theta}_1} g(\boldsymbol{x}; \Theta) = \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{\theta}_1} = \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{a}_1} \frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{\theta}_1} = \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{z}_{n-1}} \frac{\partial \boldsymbol{z}_{n-1}}{\partial \boldsymbol{a}_{n-1}} \cdots \frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{a}_1} \frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{\theta}_1} = \cdots \frac{\partial l_1(\boldsymbol{x}; \boldsymbol{\theta}_1)}{\partial \boldsymbol{\theta}_1} \tag{2.32}$$

This requires the network input $\boldsymbol{x}$ and the Jacobian of $g(\boldsymbol{x}; \Theta) = \boldsymbol{a}_n$ wrt. $\boldsymbol{a}_1$, which in turn is a product of all Jacobians $\partial \boldsymbol{a}_k / \partial \boldsymbol{z}_{k-1}$ and $\partial \boldsymbol{z}_k / \partial \boldsymbol{a}_k$ for k from n to 1.

Comparing (2.29), (2.31) and (2.32), we can see that the Jacobians wrt. the model parameters of the different layers share many expressions. This is the key to devise an algorithm that computes all of them at once. In particular, we observe:

- The Jacobian wrt. $\boldsymbol{\theta}_k$ depends on $\boldsymbol{z}_{k-1}$ (the layer's input) and on the Jacobian wrt. $\boldsymbol{a}_k$ (the layer's activation), $\partial \boldsymbol{a}_n / \partial \boldsymbol{a}_k$.
- $\boldsymbol{z}_k$ depends on $\boldsymbol{a}_k$, which in turn depends on $\boldsymbol{z}_{k-1}$. So we can compute all $\boldsymbol{z}_i$ sequentially, starting from $\boldsymbol{z}_0 = \boldsymbol{x}$.
- The Jacobian $\partial \boldsymbol{a}_n / \partial \boldsymbol{a}_k$ depends on $\partial \boldsymbol{a}_n / \partial \boldsymbol{z}_k$, which depends on $\partial \boldsymbol{a}_n / \partial \boldsymbol{a}_{k+1}$. So we can compute all Jacobians sequentially, starting from $\partial \boldsymbol{a}_n / \partial \boldsymbol{z}_{n-1}$.

Looking back to Equation 2.26, we can make another important observation: We do not explicitly need the Jacobians $\partial \boldsymbol{a}_n / \partial \boldsymbol{\theta}_k$, because we are ultimately only interested in their product with the gradient $\partial J / \partial \boldsymbol{a}_n$, to obtain the gradients $\partial J / \partial \boldsymbol{\theta}_k$.

Putting everything together, we obtain the following algorithm for computing the gradients of the penalty function wrt. all model parameters:

1. Starting from $\boldsymbol{z}_0 = \boldsymbol{x}$, compute $\boldsymbol{a}_1 = l_1(\boldsymbol{z}_0; \boldsymbol{\theta}_1)$ and $\boldsymbol{z}_1 = \phi_1(\boldsymbol{a}_1)$. Proceed in this way up until $\boldsymbol{a}_n = \boldsymbol{a}$. This is called the *forward pass*, since it propagates data through the network from input to output.
2. Compute the gradient $\partial J / \partial \boldsymbol{a}$, which often has a simple form such as $(\boldsymbol{y} - \boldsymbol{t})^T$.
3. Starting from $\delta_n = \partial J / \partial \boldsymbol{a}$, compute the gradient $\partial J / \partial \boldsymbol{\theta}_n$ by multiplication of δ_n with the Jacobian $\partial \boldsymbol{a}_n / \partial \boldsymbol{\theta}_n$ (which may depend on $\boldsymbol{z}_{n-1}$), and compute the gradient $\partial J / \partial \boldsymbol{z}_{n-1}$ by multiplication of δ_n with the Jacobian $\partial \boldsymbol{a}_n / \partial \boldsymbol{z}_{n-1}$ (which may also depend on $\boldsymbol{z}_{n-1}$). Finally, multiply the latter with the Jacobian of the activation function $\partial \boldsymbol{z}_{n-1} / \partial \boldsymbol{a}_{n-1}$ (which may depend on $\boldsymbol{a}_{n-1}$) to obtain $\delta_{n-1} = \partial J / \partial \boldsymbol{a}_{n-1}$. Proceed in this way down until $\partial J / \partial \boldsymbol{\theta}_1$. This is called the *backward pass*, since it propagates gradients through the network from output to input. Note that the gradient δ_0 wrt. the actual network input $\boldsymbol{x}$ is not required for the gradients wrt. Θ, but also easy to compute.

This is called the *error backpropagation* algorithm, or *backprop*. It has been derived by Dreyfus (1962), implemented in its current form by Linnainmaa (1970), and popularized for training neural networks by Rumelhart et al. (1986).

Note that it lends itself to a modular implementation. For each transfer function $\boldsymbol{z} = \phi(\boldsymbol{a})$, we only need two operations: (1) computing $\boldsymbol{z}$ from $\boldsymbol{a}$, and (2) computing $\Delta \boldsymbol{a}$ from $\boldsymbol{a}$ and $\Delta \boldsymbol{z}$ (i.e., multiplication with the Jacobian $\partial \boldsymbol{z} / \partial \boldsymbol{a}$), where $\Delta \boldsymbol{x}$ denotes the gradient of some scalar function wrt. $\boldsymbol{x}$ (e.g., $\partial J / \partial \boldsymbol{x}$). Similarly, for each linear transformation $\boldsymbol{a} = l(\boldsymbol{z}; \boldsymbol{\theta})$, we need three operations: (1) computing $\boldsymbol{a}$ from $\boldsymbol{z}$, (2) computing $\Delta \boldsymbol{z}$ from $\boldsymbol{z}$ and $\Delta \boldsymbol{a}$, and (3) computing $\Delta \boldsymbol{\theta}$ from $\boldsymbol{z}$, $\Delta \boldsymbol{a}$.

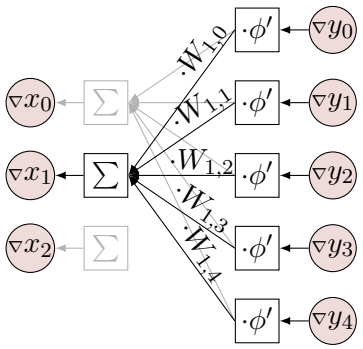


Figure 2.15: Backpropagate through a dense layer: Multiply by the derivative of ϕ, then by the transposed weight matrix.

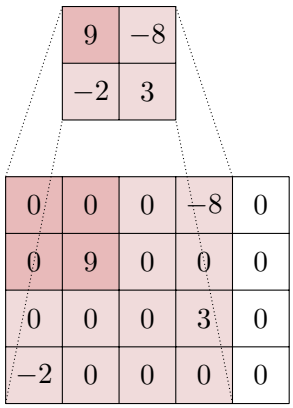


Figure 2.16: Backpropagate through max-pooling: Copy gradient to the positions of the input maxima (cf. Figure 2.12, p. 21).

To finish this section, let us see what these operations look like for the transfer functions and linear transformations discussed in Sections 2.2.2 and 2.2.3. The forward operations (computing an output given the input) are known, the backward operations (computing a gradient wrt. the input given a gradient wrt. the output and the input from the forward pass) are defined by the Jacobians.

Except for the softmax, the transfer functions $\boldsymbol{z} = \phi(\boldsymbol{a})$ process the input vector elementwise as $z_i = \phi(a_i)$. Thus, their Jacobian is a diagonal matrix of the derivatives $\partial\phi(a_i)/\partial a_i = \phi'(a_i)$, and the backward pass is an elementwise multiplication $(\Delta\boldsymbol{a})_i = (\Delta\boldsymbol{z})_i\phi'(a_i)$. In particular, the derivatives are as follows:

linear activation: For $\phi(a_i) = a_i$, we have $\phi'(a_i) = 1$, so the backward operation passes the gradient $\Delta\boldsymbol{z}$ through unchanged.

sigmoid activation: For $\phi(a_i) = \sigma(a_i) = (1+\exp(-a_i))^{-1}$, we get $\phi'(a_i) = \sigma(a_i)(1-\sigma(a_i)) = z_i(1-z_i)$ (Bishop, 2006, Eq. 4.88). Thus, the backward pass does an elementwise multiplication of the gradient with the derivative of the sigmoid.

linear rectifier: For $\phi(a_i) = \max(a_i, 0) = a_i[a_i > 0]$, we have $\phi'(a_i) = [a_i > 0]$, where $[\mathcal{A}]$ denotes the Iverson bracket that evaluates to 1 when $\mathcal{A}$ holds and to 0 otherwise. Thus, the backward pass zeros out the gradient $\Delta\boldsymbol{z}$ where the input $\boldsymbol{a}$ is nonpositive, and passes it on unchanged otherwise. Technically, $\phi'(a_i)$ is undefined at $a_i = 0$, but in practice, defining $\phi'(0) := 0$ works fine.

leaky rectifier: For $\phi(a_i) = \max(a_i, 0.01a_i) = [a_i > 0]a_i + 0.01[a_i \leq 0]a_i$, we have $\phi'(a_i) = [a_i > 0] + 0.01[a_i \leq 0]$, so the backward pass multiplies the gradient $\Delta\boldsymbol{z}$ by 0.01 where the input $\boldsymbol{a}$ is nonpositive, and passes it on unchanged otherwise. Again, defining $\phi'(0) := 0.01$ works around the discontinuity at 0.

The softmax function will only be used as an output nonlinearity along with the categorical cross-entropy, so we do not need its Jacobian in isolation.

For the linear transformations, we have full-blown Jacobians, but the multiplication with these Jacobians is reasonably easy (also see Figures 2.15 and 2.16):

dense layer: For $\boldsymbol{a} = \boldsymbol{b} + \boldsymbol{W}^T \boldsymbol{z}$, the Jacobian of $\boldsymbol{a}$ wrt. $\boldsymbol{z}$ is simply $\boldsymbol{W}^T$. Thus, the backward pass of the error is a multiplication with the transposed weight matrix: $\Delta \boldsymbol{z} = (\Delta \boldsymbol{a}) \boldsymbol{W}^T$. The Jacobian wrt. $\boldsymbol{b}$ is the identity matrix, so the gradient $\Delta \boldsymbol{b}$ is equal to the gradient $\Delta \boldsymbol{a}$. The Jacobian wrt. $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ is a 3-tensor $\boldsymbol{V} \in \mathbb{R}^{n \times m \times n}$ with $V_{i,j,k} = \partial a_i / \partial W_{j,k} = [i = k] z_j$. The product $\Delta \boldsymbol{W} = (\Delta \boldsymbol{a}) \boldsymbol{V}$ simplifies to the outer product $\Delta \boldsymbol{W} = \boldsymbol{z} (\Delta \boldsymbol{a})^T$.

convolutional layer: For $\boldsymbol{A}_j = \boldsymbol{b} + \sum_i \boldsymbol{Z}_i * \boldsymbol{W}_{i,j}$, a stack of summed 2D convolutions, it is not instructive to write out the Jacobians; we will look at the multiplication by the Jacobians only. The backward pass of the error turns out to be a stack of summed 2D convolutions $\Delta \boldsymbol{Z}_i = \sum_j \mathrm{pad}(\Delta \boldsymbol{A}_j) * \mathrm{flip}(\boldsymbol{W}_{i,j})$, where $\mathrm{pad}(\cdot)$ denotes padding the matrix with zeros ($h - 1$ rows on either side and $w - 1$ columns on either side for a filter size of $h \times w$) and $\mathrm{flip}(\cdot)$ denotes reversing the order of rows, then the order of columns. The gradient $\Delta \boldsymbol{b}$ computes as $\Delta \boldsymbol{b}_i = \sum_{j,k} \Delta \boldsymbol{A}_{i,j,k}$. Finally, the gradient $\Delta \boldsymbol{W}$ consists of convolutions of $\boldsymbol{Z}$ with $\Delta \boldsymbol{A}$: $\Delta \boldsymbol{W}_{i,j} = \boldsymbol{Z}_i * \Delta \boldsymbol{A}_j$.

pooling layer: A pooling layer can be expressed as $A_{i,j} = p\left(\boldsymbol{Z}_{ih:ih+h,\, jw:jw+w}\right)$ – each output element is a pooling function $p(\boldsymbol{S})$ applied to a $h \times w$ input submatrix $\boldsymbol{S}$. Conversely, the backward pass can be expressed in terms of these submatrices: $\Delta \boldsymbol{S} = \Delta A_{i,j}\, \partial p(\boldsymbol{S}) / \partial \boldsymbol{S}$. That is, we obtain a patch $\Delta \boldsymbol{S}$ of the gradient $\Delta \boldsymbol{Z}$ by backpropagating the corresponding element of the gradient $\Delta \boldsymbol{A}$ through the pooling function p.
For mean-pooling, we have $p(\boldsymbol{S}) = \frac{1}{hw} \sum_{k,l} S_{k,l}$, resulting in $\Delta S_{k,l} = \Delta A_{i,j} \frac{1}{hw}$. That is, the gradient is propagated to all elements of the corresponding patch, scaled by $\frac{1}{hw}$. For max-pooling, we have $p(\boldsymbol{S}) = \max_{k,l}(S_{k,l})$. This results in $\Delta S_{k,l} = \Delta A_{i,j}\, [(k,l) = \arg\max_{k,l}(S_{k,l})]$. That is, the gradient is propagated to the maximum element of the corresponding patch, since it is the only element affecting the output (infinitesimal changes to other elements do not change the output, and hence receive a gradient of 0). Similar to the rectifier on the previous page, the gradient is mathematically undefined when $\boldsymbol{S}$ has multiple elements that are maximal. In practice, this is solved either by propagating to all maximal elements, or to an arbitrary maximal element.

Equipped with the necessary tools to compute the gradient of the loss function with respect to the parameters of a deep neural network, we can now proceed to the actual optimization methods. Knowing the backpropagation mechanics in detail will later help us understand specific problems arising in the optimization.

2.2.4.3 Stochastic Gradient Descent and Variants

The backpropagation algorithm allows us to compute the gradient of the loss function $L(\Theta; f, \mathcal{D})$ with respect to the model parameters Θ – i.e., a linear approximation of the behaviour of the loss as a function of the model parameters. To reduce the loss, we merely have to change each parameter value in the opposite direction of the gradient: A parameter value with positive gradient needs to be reduced, a value with negative gradient needs to be increased. As the dependency of the loss on the parameters is not actually linear, the linear approximation only holds locally at the position the gradient was evaluated. Thus, without further information, we can only modify the parameters by small amounts, and then have to re-evaluate the gradient at the new position. Repeating this, we can move through parameter space until we reach a point of zero gradient (a local extremum or saddle point). In the following, we will discuss how different algorithms choose by how much and in which direction to change the parameters in each step, employing different ideas to optimize the loss more efficiently and escape saddle points or poor local minima.

Gradient Descent: Given a nonzero gradient, there are infinitely many directions in parameter space the loss function decreases – for example all those that move each parameter against the sign of the corresponding gradient. Gradient Descent (Cauchy, 1847) chooses the direction the loss function decreases the fastest. As it turns out, this is simply the direction of the negative gradient (illustrated in Figure 2.17, derived in Goodfellow et al., 2016, Eq. 4.3 ff.). The step size is chosen proportional to the gradient magnitude, so the update for a parameter tensor $\boldsymbol{\theta} \in \Theta$ becomes

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \Delta \boldsymbol{\theta}, \tag{2.33}$$

where η is referred to as the *learning rate*. It controls by how much to change parameters in each step, and has to be chosen outside of the optimization procedure – this is referred to as a *hyperparameter*.
The learning rate is critical: A larger learning rate moves faster through parameter space, but for a function of large curvature (or otherwise nonlinear behaviour), setting it too large may cause oscillations or divergence. Figure 2.18 illustrates this for $f(x) = x^2$. With $0.5 < \eta < 1.0$, optimization will jump back and forth over the minimum and only converge slowly. With $\eta > 1.0$, it will even move farther and farther away from the minimum. A small enough learning rate avoids this, but setting it too small slows down optimization.

Stochastic Gradient Descent: With the loss L defined as a sum of penalties J over the training set (Equation 2.4, p. 9), each update step requires a forward and backward pass of all the training data to evaluate the gradient. Since steps are generally small, we may need thousands of steps to converge, so for large train-

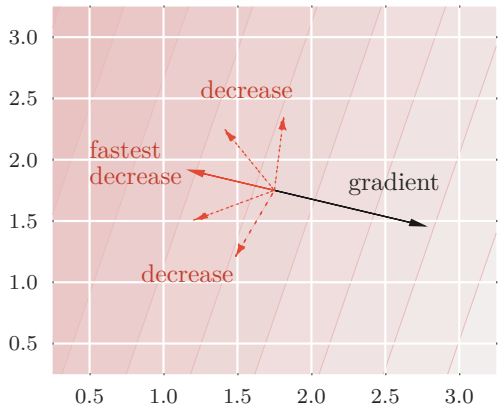


Figure 2.17: For a function of two or more arguments, a nonzero gradient at an evaluation point implies infinitely many directions the function decreases. It decreases the fastest in opposite direction of the gradient.

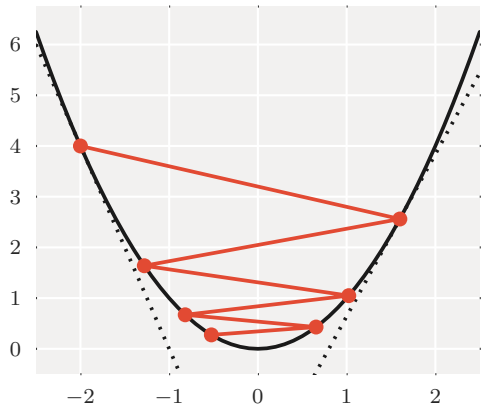


Figure 2.18: The gradient gives a local, linear approximation of a function (dotted lines). For $f(x) = x^2$, repeatedly taking too large steps in direction of the negative gradient causes oscillation over the minimum $x = 0$.

ing sets, this becomes prohibitively expensive. Stochastic Gradient Descent (SGD) defines the loss as a sum of penalties for a *mini-batch*, a subset of k data points chosen randomly for each update step. Again, k is a hyperparameter: The smaller, the faster the gradient evaluation, but the noisier the gradient estimate – i.e., the larger the expected deviation from computing the gradient on the full training set. Smaller k thus generally require smaller η to avoid taking a large step in an inaccurate direction. However, smaller k can also help optimization escape poor local minima or saddle points (Keskar et al., 2017): Noisy gradients induce exploration of the parameter space around the trajectory that would be followed by Gradient Descent on the full training set. Put differently, local minima or saddle points are unlikely to be stationary (zero-gradient) points for every mini-batch, causing optimization to search surrounding regions for a gradient to follow.
All remaining optimization methods in this section are commonly used with mini-batches. For simplicity, I will not make this explicit, but whenever referring to the gradient of the loss function, in practice it will be estimated from a mini-batch.

Momentum: As shown in Figure 2.18, Gradient Descent (or SGD) requires a low enough learning rate to avoid oscillations or divergence. While it is easy to find a suitable learning rate for $f(x) = x^2$, for more complex functions, this can

be tricky. Figure 2.19 demonstrates this for the two-dimensional Rosenbrock function $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$. It is shaped like a ravine, with steep edges in one direction, and a very small gradient in the perpendicular direction. As for the parabola, a too large learning rate causes oscillation between the two sides (Figure 2.19a). Choosing the learning rate small enough to avoid oscillations translates into slow progress along the ravine once the bottom is reached (Figure 2.19b). Momentum uses the history of gradients to accelerate progress in stable directions, and reduce progress in directions the gradient changed. Specifically, updates are computed as: [3]

$$v_{\boldsymbol{\theta}} \leftarrow \alpha\, v_{\boldsymbol{\theta}} - (1 - \alpha)\, \eta\, \Delta\boldsymbol{\theta} \tag{2.34}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + v_{\boldsymbol{\theta}} \tag{2.35}$$

Here, $v_{\boldsymbol{\theta}}$ is an exponential moving average over the gradients $\Delta\boldsymbol{\theta}$, with coefficient α as an additional hyperparameter. It can be interpreted as the velocity of a sphere moving through parameter space, with friction controlled by α, force exerted by the gradient, and simulation step size determined by η. In our example, this slightly dampens the frequently changing gradient across the ravine, and builds up velocity along the perpendicular direction that has a small but steady gradient (Figure 2.19c). Momentum can also help escape local minima and saddle points, by continuing in the direction of $v_{\boldsymbol{\theta}}$.

Nesterov Accelerated Gradient: As a slight modification of Momentum, Nesterov Accelerated Gradient (NAG) evaluates the gradient for the velocity update not at the current position $\boldsymbol{\theta}$ in parameter space, but at the position $\boldsymbol{\theta}'$ we would reach with the current velocity scaled by α:

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} + \alpha\, v_{\boldsymbol{\theta}} \tag{2.36}$$

$$v_{\boldsymbol{\theta}} \leftarrow \alpha\, v_{\boldsymbol{\theta}} - (1 - \alpha)\, \eta\, \Delta\boldsymbol{\theta}' \tag{2.37}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + v_{\boldsymbol{\theta}} \tag{2.38}$$

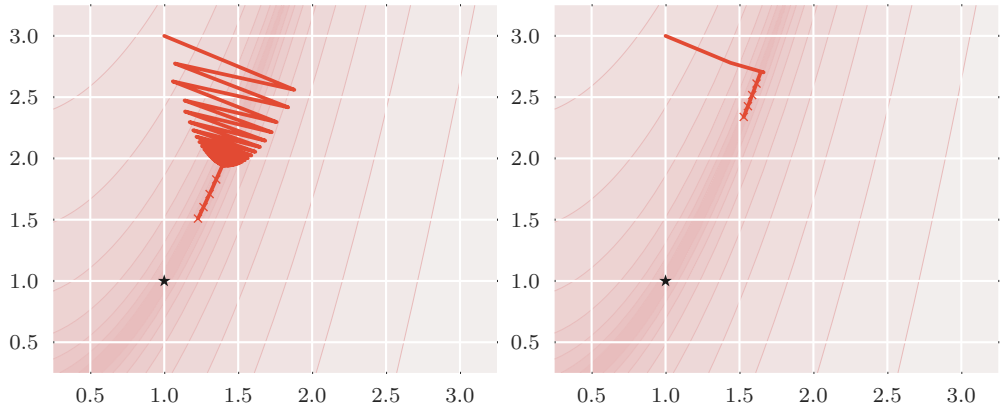
This formulation was derived by Sutskever et al. (2013) from the original proposal by Nesterov (1983). Bengio et al. (2013, Sec. 3.5) further reformulated it to evaluate the gradient at the current position and employ different updates:

$$v_{\boldsymbol{\theta}} \leftarrow \alpha\, v_{\boldsymbol{\theta}} - (1 - \alpha)\, \eta\, \Delta\boldsymbol{\theta} \tag{2.39}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, v_{\boldsymbol{\theta}} - (1 - \alpha)\, \eta\, \Delta\boldsymbol{\theta} \tag{2.40}$$

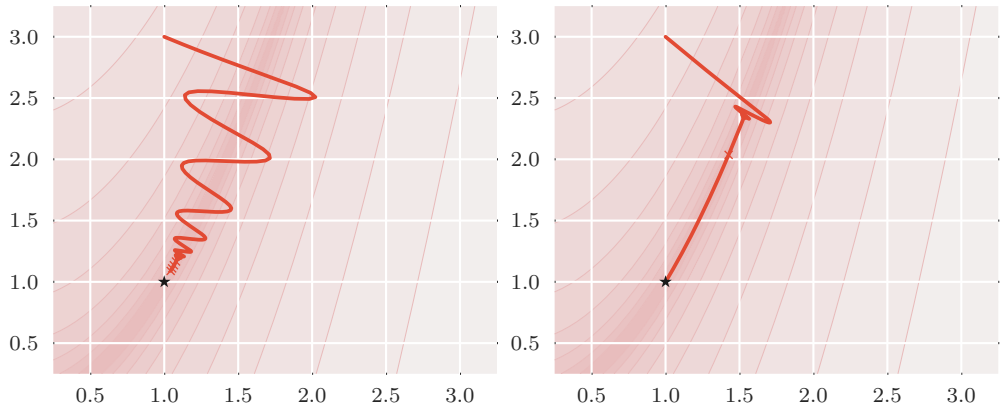
Compared to Momentum, the "anticipatory" gradient evaluation in (2.37) or, equivalently, the stronger reliance on the current gradient over the velocity in (2.40) dampens oscillations more quickly (not included in Figure 2.19).

[3] The velocity update is often formulated without the $(1 - \alpha)$ term in Equation 2.34 – but this induces a strong interaction between η and α, and is inconsistent with ADAM described on p. 32. Experiments in this thesis include the $(1 - \alpha)$ term whenever training with varying α.



(a) Gradient Descent, $\eta = 1.1\text{e}{-3}$

(b) Gradient Descent, $\eta = 5.5\text{e}{-4}$



(c) Momentum, $\eta = 1.1\text{e}{-3}$, $\alpha = 0.95$

(d) ADAM, $\eta = 5.5\text{e}{-2}$, $\beta_1 = 0.9$, $\beta_2 = 0.99$

Figure 2.19: Behaviour of different gradient-based optimization algorithms near a ridge (here: the Rosenbrock function $(1 - x)^2 + 100(y - x^2)^2$). Each graph shows the trajectory of up to 2000 optimization steps starting at the same position, with small crosses every 500 steps. Contour lines are spaced logarithmically, and the minimum is marked by a star.
(a) Gradient Descent may initially oscillate between the steep sides.
(b) Halving the learning rate avoids the oscillation, but also slows down progress along the bottom of the ridge, where gradients are smaller.
(c) With momentum, the oscillation is slightly dampened, and the optimizer builds up velocity along the ridge towards the minimum.
(d) ADAM scales steps by a long-term exponential moving average of the gradient magnitudes, requiring a different learning rate. It quickly dampens oscillations between the steep sides and accelerates inside the ridge, reaching the minimum in under 1000 steps.

ADAM: While Momentum and NAG can dampen oscillations and amplify weak, but steady gradients, progress still depends on the gradient magnitude. For functions of very different gradient magnitudes – such as the Rosenbrock function in Figure 2.19 – this is a serious limitation we cannot overcome by tuning the hyperparameters η and α. To illustrate, reconsider Figure 2.19c. Even with Momentum, progress slows down considerably once the bottom of the ridge is reached. In this phase, increasing η (or, to some extent, α) would accelerate convergence, but increasing it from the beginning would lead to heavier oscillations or divergence before reaching the ridge's bottom.
Adaptive Moment Estimation (ADAM) by Kingma and Ba (2015) solves this by computing a long-term exponential moving average of the gradient magnitudes in each dimension, and dividing the updates by it. This way, updates are amplified (or scaled down) in regions and dimensions of notoriously small (or large) gradients, ensuring steady progress even in functions of very inhomogeneous gradient magnitudes. The update rules become:

$$v_{\boldsymbol{\theta}} \leftarrow \beta_1 \, v_{\boldsymbol{\theta}} - (1 - \beta_1) \, \Delta\boldsymbol{\theta} \tag{2.41}$$

$$m_{\boldsymbol{\theta}} \leftarrow \beta_2 \, m_{\boldsymbol{\theta}} - (1 - \beta_2) \, (\Delta\boldsymbol{\theta})^2 \tag{2.42}$$

$$t \leftarrow t + 1 \tag{2.43}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \, \frac{v_{\boldsymbol{\theta}}}{\sqrt{m_{\boldsymbol{\theta}}} + \epsilon} \, \sqrt{1 - \beta_2^t} \, / \, (1 - \beta_1^t) \tag{2.44}$$

As before, $v_{\boldsymbol{\theta}}$ is a running average of gradients. $m_{\boldsymbol{\theta}}$ is the running average of element-wise squared gradients, with its own decay coefficient. t, $v_{\boldsymbol{\theta}}$ and $m_{\boldsymbol{\theta}}$ are all initialized to zero(s). Near the beginning of training, $v_{\boldsymbol{\theta}}$ and $m_{\boldsymbol{\theta}}$ are thus much smaller than $\Delta\boldsymbol{\theta}$ and $(\Delta\boldsymbol{\theta})^2$, especially with decay coefficients β_1 and β_2 close to 1. The term $\sqrt{1 - \beta_2^t} \, / \, (1 - \beta_1^t)$ counters this – it rescales $v_{\boldsymbol{\theta}}$ and $m_{\boldsymbol{\theta}}$ into unbiased estimates of the first moment and raw second moment of the gradient, respectively. Typical coefficients are $\beta_1 = 0.9$ and $\beta_2 = 0.999$, so the magnitude estimates are much more long-term than the velocities. ϵ is used to avoid extreme scalings or a division by zero; Kingma and Ba (2015) suggest $\epsilon = 1\text{e}{-8}$. The division by the gradient magnitudes makes the update invariant to rescaling the objective function or the gradients. Thus, ADAM usually needs a different learning rate than SGD, Momentum or NAG, which are sensitive to scale. In our Rosenbrock function test case, with a suitable learning rate, ADAM progresses at a steady pace both downwards the steep edges and along the flat bottom of the ridge (Figure 2.19d).

We have now discussed the most common optimization schemes in contemporary deep learning. All of these rely on first-order (gradient) information only, using sophisticated heuristics to determine how far to follow the negative gradient in each

step. This begs the question why algorithms do not employ second-order (curvature) information – since low curvature implies a stable gradient, this could directly be used to determine an appropriate step size. In fact, around 2010, considerable effort was spent on efficiently using second-order derivatives in optimizing deep neural networks, e.g., Hessian-Free Learning (Martens, 2010) or Krylov Subspace Descent (Vinyals and Povey, 2012). However, this line of research became less important after progress in a previously neglected part of optimization we will discuss in the following section.

2.2.4.4 Initialization

For nonconvex functions – i.e., functions with multiple extrema, not just a single minimum – gradient-based optimization schemes will generally converge to a local minimum that is not globally optimal. Which minimum they converge to does not only depend on the chosen scheme and its hyperparameters, but also on the starting point in parameter space. For a particularly bad choice of the starting position, such as a saddle point, gradient-based optimization may not even be able to proceed at all. To understand what constitutes a bad choice – and, conversely, how to do better – we need to reconsider how we compute gradients for a deep network.

Recalling the backpropagation algorithm (Section 2.2.4.2, p. 23), we first compute the prediction of the network for a training example (the forward pass). We then compute the gradient of the penalty function J with respect to the network output – this gives us the desired change of the output to reduce the loss. Starting with the output layer and going backwards, we multiply this gradient by the Jacobians of all layers that were involved in the forward pass, to obtain the desired change of each layer's input and weights. The deeper our model, the more layers we have, and the more Jacobians the gradient will be multiplied with. Both for dense and for convolutional layers, the multiplication by the Jacobian uses the same parameter tensors as in the forward pass. Thus, the initial model parameters do not only affect the initial forward pass behaviour, but also the backward pass – i.e., the gradient with respect to the weights. Let us now consider which initial conditions lead to gradients that hinder optimization.

When using the sigmoid transfer function (Equation 2.10, p. 16), the backward pass multiplies the gradient by the function's derivative $z(1 - z)$, with z denoting the output of the sigmoid function in the forward pass (see p. 26). This value is small when z is close to zero or close to one, i.e., when the input to the sigmoid has a large magnitude (this is also visible from Figure 2.8a, p. 17: the slope of the sigmoid gets small near both ends). This effect is called *saturation* of the sigmoid. For optimization to work, the initial network must not saturate sigmoid units – otherwise the gradient wrt. weights into these units, and wrt. weights of all layers before these units, will be driven close to zero.

Even with nonsaturating transfer functions such as the linear rectifier (Equation 2.13, p. 16), the multiplied Jacobians of all linear transformations may culminate in a matrix of very small or very large values, resulting in very small or very large gradients for layers far from the output. Both hinders learning, and becomes more severe with increased model depth (and within a deep model, it affects layers close to the input more strongly than layers close to the output).

Given these considerations, it becomes clear that for deep models, only a very limited region in parameter space provides starting conditions for successful gradient-based optimization. Early attempts at training deep models from random initial weights have thus been unsuccessful. Only since 2006, different strategies have been found that provide initial model parameters allowing optimization:

Pretraining: Considered a breakthrough in deep learning, Hinton et al. (2006) showed that deep models could be trained after unsupervised, layer-wise *pretraining*. Starting with a two-layer graphical model called a Restricted Boltzmann Machine (RBM) that learns a generative model of the input distribution, then successively training additional RBMs each on the previous RBM's latent representation, one obtains a sequence of weight matrices and bias vectors to initialize a deep MLP. This network can then be trained supervisedly with any variant of SGD – in this context, this is termed *fine-tuning*. The intuition behind this procedure is to use unsupervised learning to discover a general-purpose feature hierarchy describing the data, then tuning it to solve a particular task. Bengio et al. (2007) demonstrated that the same principle was possible with auto-encoders in place of RBMs – shallow MLPs that learn to reconstruct their own output through nonlinear transformations.

Thoughtful scaling: Glorot and Bengio (2010) investigated why optimization of deep models from random initialization would fail in the first place. They identified the causes discussed above: saturation of sigmoid units – calling for small initial weights as a countermeasure – and vanishing gradients in backpropagation, due to too small initial weights (and therefore, small Jacobians). They proposed to initialize weights randomly, but scaled exactly such that each layer would retain the variance of input data in the forward pass, and the variance of gradients in the backward pass. For a dense layer $D(\boldsymbol{x}; \Theta, \phi) = \phi(\boldsymbol{b} + \boldsymbol{W}^T \boldsymbol{x})$ with linear activation $\phi(\boldsymbol{a}) = \boldsymbol{a}$, weight matrix $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ and i.i.d. random W_{ij} and x_i, asserting $\mathrm{Var}\left[D(\boldsymbol{x})_j\right] = \mathrm{Var}\left[x_i\right]$ results in the requirement $\mathrm{Var}\left[W_{ij}\right] = 1/n$. Similarly, for the backward pass, asserting $\mathrm{Var}\left[\Delta x_i\right] = \mathrm{Var}\left[\Delta D(\boldsymbol{x})_j\right]$ results in $\mathrm{Var}\left[W_{ij}\right] = 1/m$. These can only be simultaneously fulfilled for $m = n$, i.e., layers of as many output as input units. For the general case, Glorot and Bengio (2010, Eq. 12) suggest

$$\mathrm{Var}\left[W_{ij}\right] = \frac{2}{n+m}. \tag{2.45}$$

Specifically, they propose to draw weights from the uniform distribution

$$W_{ij} \sim U\Big(-\frac{\sqrt{6}}{\sqrt{n+m}}, \frac{\sqrt{6}}{\sqrt{n+m}}\Big), \tag{2.46}$$

which fulfils (2.45) since $\operatorname{Var}[r] = (b-a)^2/12$ for $r \sim U(a,b)$. This scheme can also be applied to convolutional layers with $\boldsymbol{W} \in \mathbb{R}^{a\times b\times c\times d}$: it requires $n = acd$ as the *fan-in* (number of inputs per forward pass output) and $m = bcd$ as the *fan-out* (number of inputs per backward pass output).
He et al. (2015, Sec. 2.2) extend the analysis to the rectifier $\phi(\boldsymbol{a}) = \max(\boldsymbol{a}, \boldsymbol{0})$, obtaining requirements $\operatorname{Var}[Wij] = 2/n$ for the forward and $\operatorname{Var}[Wij] = 2/m$ for the backward pass. They suggest to draw weights from the Gaussian distribution

$$W_{ij} \sim \mathcal{N}\Big(0, \frac{2}{n}\Big). \tag{2.47}$$

This violates the backward pass requirement when $n \neq m$, but for common CNN architectures, this does not happen often enough to become problematic. Combined with biases initialized to zero, either of these methods has been used successfully to train deep neural networks.

Orthogonal matrices: Saxe et al. (2014) investigated why even with carefully-scaled random initialization, convergence is slower than with unsupervised pretraining. They found that while the Jacobian $\partial f(\boldsymbol{x})/\partial \boldsymbol{x}$ of a randomly-initialized network $\boldsymbol{y} = f(\boldsymbol{x})$ preserves the variance of a random gradient $\Delta \boldsymbol{y}$ when weights are carefully scaled (e.g., following Equation 2.46), its singular value distribution is highly skewed: most singular values are very small, while some are very large. Thus, the Jacobian attenuates most directions of variance, countered by strong amplification along a small portion of singular vectors. In contrast, unsupervised pretraining yields initial conditions with a much more uniform distribution of singular values. To obtain similar conditions, Saxe et al. propose to initialize weights as scaled random orthogonal matrices:

$$\boldsymbol{W} = s\,\boldsymbol{U} \quad \text{s.t. } \boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I} \tag{2.48}$$

This way, all singular values are exactly 1 both for a single weight matrix and for a product of multiple such matrices. For $n < m$ (or $n > m$), one would use the first n rows (or first m columns) of an orthogonal matrix. A random orthogonal matrix can be generated by computing the Singular Value Decomposition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$ of a random matrix $\boldsymbol{X}$. Depending on the nonlinearity ϕ, the scale s needs to be chosen to preserve the variance in the forward or backward pass: For a linear layer, it is $s = 1$, for the rectifier, it is $s = \sqrt{2}$, and for the leaky rectifier, it is $s = \sqrt{2/(1+0.01^2)}$.

Data-dependent scaling: All random initializations discussed above employ theoretical arguments for determining the scales of initial weights. These are based on assumptions about the data distribution, nonlinearities and network architecture that may not hold in all cases encountered in practice. As an easy way out, several authors propose to determine the scales empirically (Krähenbühl et al., 2016; Mishkin and Matas, 2016; Salimans and Kingma, 2016): Starting with the first layer, propagate a large enough set of training examples through its linear transformation (but not yet its nonlinearity, if any), determine the standard deviation of the transformed data and divide the weights by it. Continue with the following layer and repeat until all layers have been processed. This scheme is applicable to any nonlinearity and layer type, and can correct the scales for any type of random initialization to ensure the variance of inputs is preserved in the forward pass.

Finally, we have all the necessary ingredients to optimize deep neural networks on a training set: Appropriate loss functions (Section 2.2.4.1), methods to compute the gradient of (Section 2.2.4.2) and minimize the loss (Section 2.2.4.3), and initialization procedures for the model parameters to be optimized (Section 2.2.4.4).

2.2.5 Generalization

As for any machine learning method, our goal is not to obtain a model that performs optimally on the training data, but one that generalizes well to unseen examples. In deep learning, this can be particularly challenging: Since deep neural networks often have millions of learnable parameters, they are prone to overfitting (Section 2.1.3) – finding variations in the training data that help predict the training labels correctly, but do not carry over to examples outside the training set. As for optimization, deep learning research has adopted, adapted and invented a range of methods to improve generalization:

Adding more training data: As discussed on p. 13, increasing the amount of training examples adds more constraints on the parameters and lowers the risk of overfitting. Due to the size of modern deep networks, deep learning has a reputation of requiring very large training sets, and indeed some of the successes in deep learning would not have been possible without datasets collected via the Internet. However, there are several alternative and complementary techniques to reduce overfitting without collecting more data.

Choosing a specialized model: As discussed on p. 12, defining the tunable function $f(\boldsymbol{x}; \Theta)$ to incorporate prior knowledge about the structure of the inputs, the targets or their mapping restricts the search space. With a suitable choice of f, this can rule out unwanted solutions that are unlikely to generalize. For

images and other types of data characterized by spatial or temporal relations between the input features, using convolutional layers (p. 18) is a specialization of the model that improves generalization.

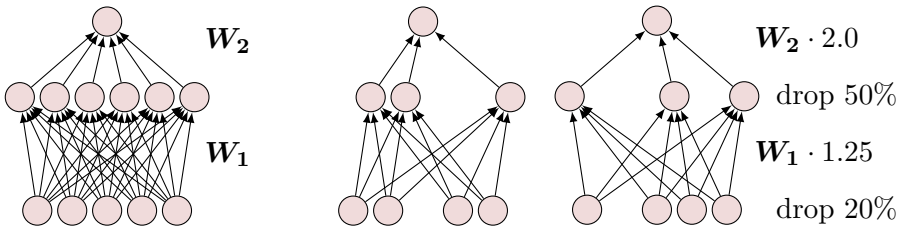
Lowering model complexity: As discussed on p. 12, reducing the number of model parameters generally reduces the risk of overfitting. Again, this is an argument for using convolutional layers: As the number of parameters only depends on the number and size of filters, not on the input size, they can be tuned to require much fewer parameters than a dense layer processing the same input tensor.

This is also an argument for favouring small convolutions: Three stacked 3×3 convolutions have the same receptive field as a single 7×7 convolution, but only $27/49 \approx 55\%$ of parameters. Simonyan and Zisserman (2015, Sec. 2.3) popularized architectures relying on 3×3 convolutions exclusively.

When not using nonlinearities in between, stacking multiple 3×3 convolutions can be seen as a factorization of a larger convolution kernel. Factorizations provide a general tool for trading model expressivity against model complexity: Szegedy et al. (2016b, Sec. 3.2) propose to factorize $h \times w$ convolutions as a $h \times 1$ and a $1 \times w$ convolution to further reduce the number of parameters. For dense layers, a $n \times m$ matrix can be factorized into a product of a $n \times k$ and a $k \times m$ matrix, with $k < \min(m, n)$, to reduce the parameter count (Sainath et al., 2013b). Novikov et al. (2015) experiment with more complex factorizations of dense layers as a series of reshape operations and matrix multiplications.

Pretraining: Unsupervised pretraining can serve as an initialization procedure to enable gradient-based optimization of very deep models, just like carefully-scaled random initializations (p. 34). But unlike random values, it learns a feature hierarchy useful for generating and describing the input data distribution. Erhan et al. (2009) showed that this serves as a strong regularizer: optimization of pretrained models passes through very different regions of parameter space and ends up in better-generalizing local minima than randomly-initialized models.

Early stopping: Neural networks are usually trained in an iterative fashion, gradually decreasing the value of the loss function for the training data until reaching a local minimum (p. 28). At any point during training, we can estimate how well it generalizes to unseen data by computing the value of the loss function for a held-out set of examples not used in training (Section 2.1.3, p. 11). Typically, the former – the *training loss* – will continually decrease over the course of training, while the latter – the *validation loss* – will initially decrease, and eventually increase again.



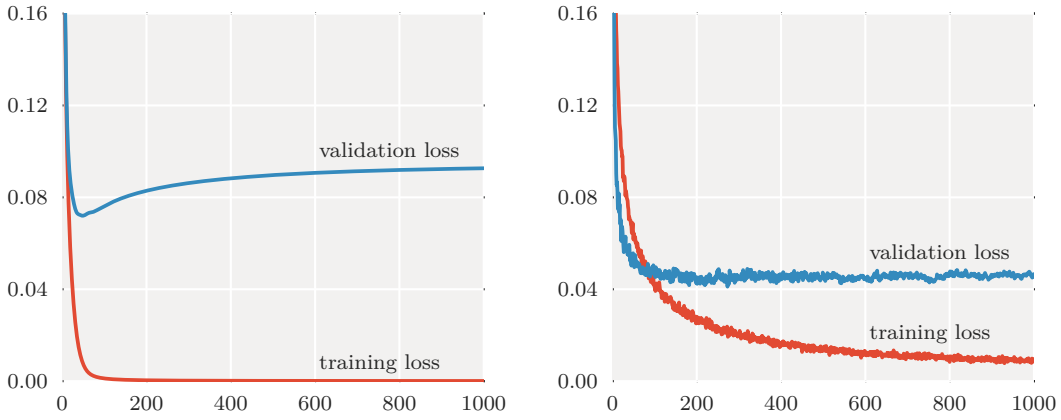
(a) original model (b) two samples of model with random dropout

Figure 2.20: Illustration of dropout for a simple MLP (a). For each training example, a random subset of units is omitted, and remaining unit activations (or, equivalently, outgoing weights) are scaled up to compensate (b). At test time, the full model is used (a).

Figure 2.21a illustrates this for an MLP trained to recognize hand-written digits. After about 50 epochs (i.e., enough iterations to cover the full set of training examples 50 times), the validation loss reaches a minimum. Training beyond this point reduces the training loss, but increases the validation loss: the network is overfitting to the training data. Early stopping means monitoring the validation error during training, and stopping training once the validation error stops improving (which may entail some heuristics to cope with more noisy loss curves as in Figure 2.21b).

Weight decay: As discussed on p. 13, limiting the magnitudes of parameters that are multiplied with the inputs results in a function f that only changes slowly over the input space. This rules out solutions that lock in exactly to the input feature combinations of training examples to reproduce the training targets. For neural networks, the parameters engaging in multiplicative interactions with inputs are the weights of dense layers and filter kernels of convolutional layers, and penalizing them with L_2 regularization (Equation 2.7, p. 13) effectively improves generalization.

Dropout: Hinton et al. (2012b) propose a regularization method specialized towards gradient-based optimization of deep neural networks: For every training example and network layer, randomly set some of the layer's inputs to zero with a given probability $p \in [0, 1)$, and scale the surviving inputs by $1/(1-p)$ to compensate (such that the expected mean over all inputs stays the same). Figure 2.20 illustrates this for a simple MLP of two layers, dropping the first layer's inputs with probability $p = 20\%$ and the second layer's inputs (i.e., the first layer's outputs) with $p = 50\%$. At test time, when computing predictions for validation or test examples, dropout probabilities p are set to zero.



(a) trained without dropout

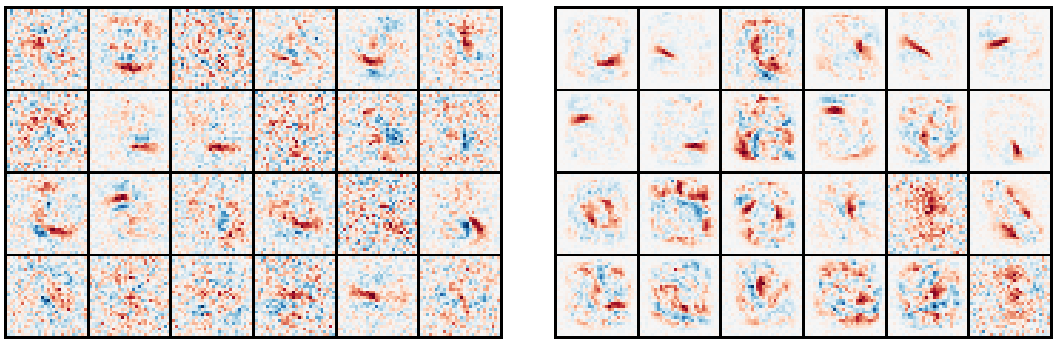
(b) trained with dropout

Figure 2.21: Training and validation loss monitored while training an MLP for digit recognition. Without dropout (a), it starts overfitting after 50 epochs. With dropout (b), it does not overfit within 1000 epochs of training.

The effect of dropout can be explained in different ways:

- A general method to improve generalization is to train multiple models (a so-called *ensemble*), possibly on different parts of the training set, and average their predictions (*bagging*). Dropout can be interpreted as training an ensemble of 2^N models (where N is the number of units with dropout probability $p > 0$), most of them on a single training example each, regularized and made feasible by sharing weights between all models. Predictions at test time are similar to averaging over the predictions of the ensemble members.[4]
- A possible way of overfitting is to find a small subset of features that happen to suffice to explain the training data, and focus on this subset exclusively. This can lead to nonrobust solutions that draw from feature constellations in the training set that do not generalize. For example, in hand-written digit recognition from images, there might be a single pixel that suffices to distinguish all eights and zeros in the training data, but not all eights and zeros in the world. Dropout prevents units from focusing on very few units in the previous layer, since they are not present in every sample of the model. This drives the network to learn more robust features taking into account larger subsets of units.

[4] For an MLP with softmax output and a single hidden layer, prediction without dropout is exactly equivalent to taking the geometric mean over all 2^N models (Hinton et al., 2012b, p. 2). For deeper MLPs, it is still a good approximation (Srivastava et al., 2014, Fig. 11).



(a) trained without dropout (b) trained with dropout

Figure 2.22: First-layer connection weights of an MLP trained for digit recognition. Each square depicts connections of input pixels towards a single hidden unit, with negative values in red and positive values in blue. Without dropout (a), features are often noisy. With dropout (b), many units connect to multiple correlated input pixels forming strokes.

- Since dropout forcefully multiplies some unit outputs with zero, these units will also not receive any gradient in the backward pass: Dropout can be seen as an elementwise multiplication of a layer's input with a binary mask (a tensor of zeros and ones), and the backward pass through this operation is an elementwise multiplication of the gradient by the same mask. Put differently, the gradient will only be distributed over the units that were not dropped. Thus, dropout ensures that all units in a layer receive gradients and adjust their connection weights from time to time, even if some of their peers already found a way to match the training data (in which case, without dropout, the loss would be zero and none of the units would receive a gradient). This drives the network to learn multiple alternative solutions to a problem, and have many units contribute to the solution rather than a few well-performing ones.

Figure 2.21 shows quantitatively how dropout improves generalization: When training an MLP for hand-written digit recognition with dropout, the training and validation losses per epoch are noisier than without dropout, but the validation loss does not increase even after 1000 epochs (note that eventually it still would). Furthermore, it achieves a better validation loss than training without dropout combined with early stopping around epoch 50. Figure 2.22 shows the qualitative effect: With dropout, units learn to connect to input pixels that are correlated, to be robust against omission of a fraction of pixels. Without dropout, only some units learn such stroke-like features, many rely on constellations of seemingly unrelated pixels spread across the input.

(a) original (b) flipped (c) rotated (d) scaled/cropped

Figure 2.23: Illustration of data augmentation: From an original training example (a), we can create additional examples using transformations (b, c, d) that either keep the label unchanged or affect it in a known way. Photograph courtesy of Bertil Videt (2006).

Data augmentation: As a variation on the idea of adding more training data, data augmentation synthesizes additional training examples from the existing training set by transforming the inputs and/or targets in a way that is consistent with the mapping to be learned. For example, for object recognition in images, we can generate additional input-target pairs by any combination of horizontally flipping an image, moderately rotating it or scaling and cropping it, while keeping the associated label unchanged (see Figure 2.23). For object localization, we need to apply the same transformations to the target locations or bounding boxes to match the transformed inputs.
The effect of this is twofold: (a) It increases the ratio of training examples to learnable parameters, without requiring additional curation of data, and (b) it allows to teach the model to become invariant or equivariant to specific transformations, providing a brute-force way to incorporate prior knowledge about the task at hands.
Note that in order to improve generalization to unseen data, the transformations have to be chosen well: If the synthesized examples do not reflect the distribution of input-target pairs underlying the task, the model will waste capacity on irrelevant regions of the input space, or learn wrong mappings. For example, unconstrained rotation in an object recognition task may teach a network to recognize upside-down images of buildings or lakes, but such images will be highly unlikely in real-world data.

While other regularization methods exist, the ones discussed above are the most important in contemporary deep learning – and in the main part of my thesis.

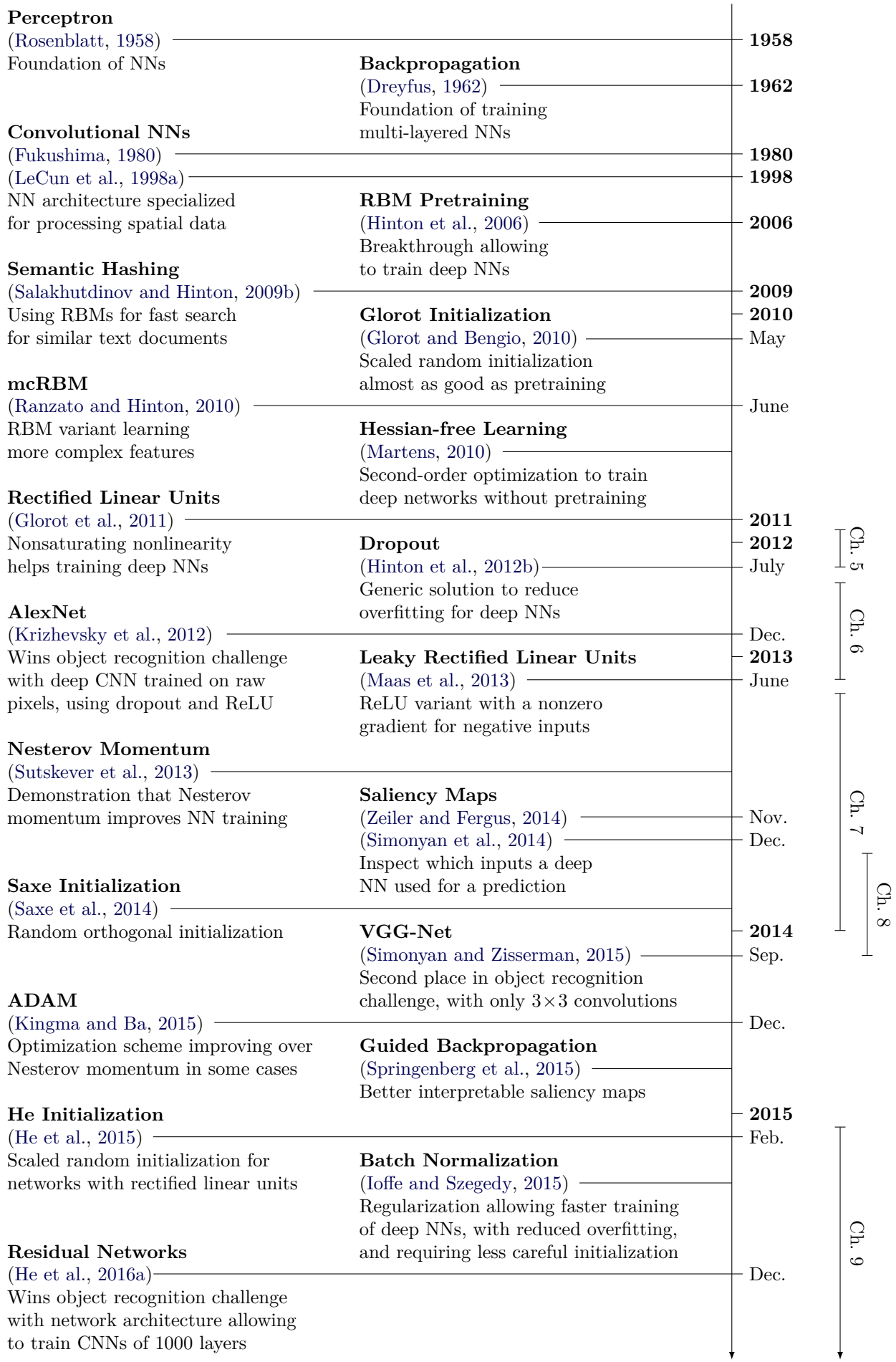
2.3 Timeline

When I started working on the research projects forming my thesis, several of the methods described in this chapter had not been invented yet, not published yet, or not widely known to work yet. To allow the reader to better understand the decisions in the main part of this thesis, the figure on the right provides a timeline with the most relevant deep learning publications for my work as time instances on the left, and my thesis chapters as time periods on the right.

For one, this overview gives some context to my work, showing which methods were known when I worked on which topic. On the other hand, it shows how quickly the field of deep learning research has moved during my thesis, and how the changes in methodology in my work reflect the learning process of the field as a whole.

Note that for each method, the timeline only gives the time point of first publication – either as a conference paper, or as a preprint (in case of a preprint, the citation still references the official peer-reviewed publication, which may be up to half a year later). After initial publication of a method, it still takes some time until it has been tried and tested by other researchers and possibly gains widespread acceptance. The timeline thus does not reflect when I became aware of a method.

Apparent gaps between chapters are due to research outside this thesis (see p. 253), research that lead to a dead end (described at the end of each chapter), or work on industry projects (a part of which are detailed in Appendix A).


Perceptron
(Rosenblatt, 1958)
Foundation of NNs
1958
Backpropagation
(Dreyfus, 1962)
Foundation of training
multi-layered NNs
1962
Convolutional NNs
(Fukushima, 1980)
1980
(LeCun et al., 1998a)
1998
NN architecture specialized
for processing spatial data
RBM Pretraining
(Hinton et al., 2006)
Breakthrough allowing
to train deep NNs
2006
Semantic Hashing
(Salakhutdinov and Hinton, 2009b)
Using RBMs for fast search
for similar text documents
2009
Glorot Initialization
(Glorot and Bengio, 2010)
Scaled random initialization
almost as good as pretraining
2010
May
mcRBM
(Ranzato and Hinton, 2010)
RBM variant learning
more complex features
June
Hessian-free Learning
(Martens, 2010)
Second-order optimization to train
deep networks without pretraining
Rectified Linear Units
(Glorot et al., 2011)
Nonsaturating nonlinearity
helps training deep NNs
2011
Dropout
(Hinton et al., 2012b)
Generic solution to reduce
overfitting for deep NNs
2012
July
Ch. 5
AlexNet
(Krizhevsky et al., 2012)
Wins object recognition challenge
with deep CNN trained on raw
pixels, using dropout and ReLU
Dec.
Leaky Rectified Linear Units
(Maas et al., 2013)
ReLU variant with a nonzero
gradient for negative inputs
2013
June
Ch. 6
Nesterov Momentum
(Sutskever et al., 2013)
Demonstration that Nesterov
momentum improves NN training
Saliency Maps
(Zeiler and Fergus, 2014)
(Simonyan et al., 2014)
Inspect which inputs a deep
NN used for a prediction
Nov.
Dec.
Ch. 7
Ch. 8
Saxe Initialization
(Saxe et al., 2014)
Random orthogonal initialization
VGG-Net
(Simonyan and Zisserman, 2015)
Second place in object recognition
challenge, with only 3×3 convolutions
2014
Sep.
ADAM
(Kingma and Ba, 2015)
Optimization scheme improving over
Nesterov momentum in some cases
Dec.
Guided Backpropagation
(Springenberg et al., 2015)
Better interpretable saliency maps
He Initialization
(He et al., 2015)
Scaled random initialization for
networks with rectified linear units
2015
Feb.
Batch Normalization
(Ioffe and Szegedy, 2015)
Regularization allowing faster training
of deep NNs, with reduced overfitting,
and requiring less careful initialization
Ch. 9
Residual Networks
(He et al., 2016a)
Wins object recognition challenge
with network architecture allowing
to train CNNs of 1000 layers
Dec.

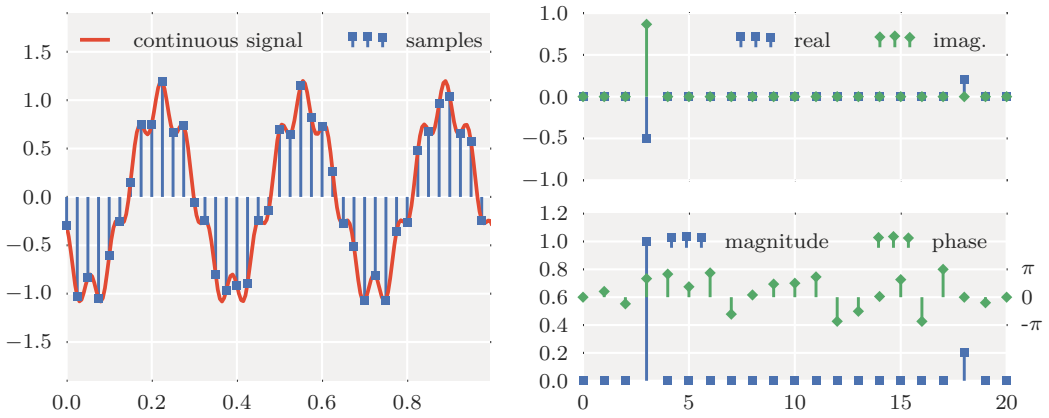
3 A Primer on Audio Signal Processing

Besides a background in deep learning, the main chapters of this thesis also assume a basic knowledge of digital signal processing given in the following. After explaining the digital representation of sound waves and a common way to analyse their frequency content over time, we will look into representations capturing aspects of human auditory perception. Finally, we will briefly discuss algorithms aiming to extract more abstract information (features) from these representations that are useful for analysing music signals. While for most purposes we will not make use of these algorithms, knowing them is useful when reasoning about how to design neural networks that process audio data.

Just as the previous chapter, this primer only covers material required to follow the thesis and omits much of the formal background. For a thorough introduction to digital signal processing, see the books of Lyons (2010) or Oppenheim and Schafer (2009), and for more focus on music processing, see Müller (2015).

3.1 From Waveform to Spectrogram

As a foundation, we will discuss how sound is commonly represented in a computer and how to compute a spectrogram, the basis for all further audio analysis methods discussed in this chapter.



(a) waveform

(b) spectrum (top: rectangular, bottom: polar)

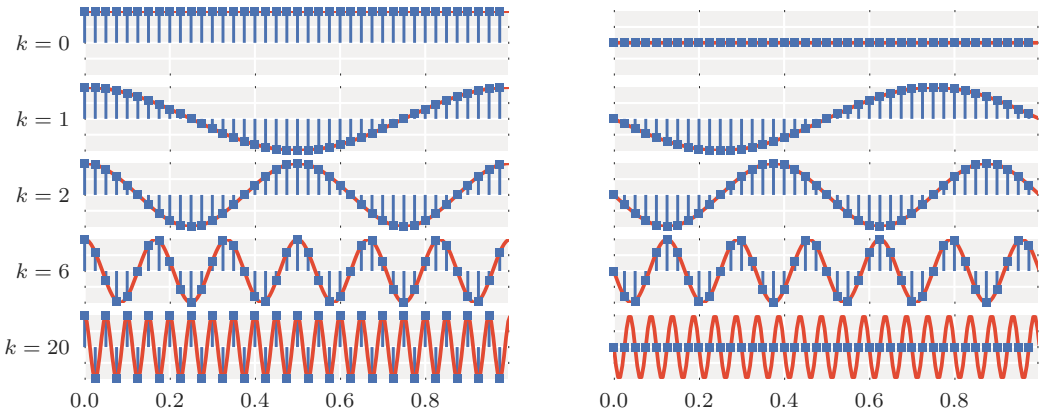
Figure 3.1: A short sound recording represented as a sequence of 40 amplitude values over time (a), or as mixing coefficients of 40 sinusoids (b).

3.1.1 Digital Sound Recording

Sound is a local variation in pressure that can travel through a medium such as air or water as a wave and be registered by any device capable of measuring variations in pressure, or local movement of the medium. Sound can be characterized by the amount of variation and the rate of variation over time, as well as the travel direction of the sound waves.

To capture the amount and rate of variation, we can record the deviation of pressure from normal pressure over time, obtaining a function as depicted in Figure 3.1a (solid red line), a *waveform* representation. To also capture the travel direction, we could use multiple such recordings from slightly different spatial locations – since sound travels at a finite speed, the distances incur a direction-dependent delay between the recordings that allows to infer the travel direction. However, as all tasks considered in this thesis do not require to know about the travel direction, we will limit ourselves to single-channel recordings.

In order to process sound with a computer, the continuous function of pressure deviation over time has to be quantized in time and amplitude to obtain a finite-length representation. Specifically, the waveform is represented as a sequence of *samples* of pressure deviation amplitudes, taken at a regular time interval, also depicted in Figure 3.1a (blue squares). The *sample rate*, the number of samples per second, determines the quantization in time. It limits the rate of amplitude variation that can be captured: With a sample rate of 40 per second (40 Hz) one can capture oscillations of up to 20 Hz, as depicted in Figure 3.2a, bottom row.



(a) cosine templates (b) negative sine templates

Figure 3.2: Basis functions used to compute the spectrum in Figure 3.1b: Cosine functions for the real part (a), sine functions for the imaginary part (b)

Any oscillations faster than this so-called *Nyquist rate* would appear as oscillations with a lower frequency when sampled, and are thus usually filtered out by analogue circuits as part of the recording setup. Typical sample rates in audio recordings are 22,050 Hz or 44,100 Hz, capturing oscillations up to 11,025 Hz or 22,050 Hz, respectively. Amplitude values are usually quantized as 16-bit integers or 32-bit floating point. This quantization incurs some noise, but can be ignored for the purposes of this thesis.

3.1.2 Time Domain and Frequency Domain

At a sample rate of 44.1 kHz, even a short sound recording has a lot of data points. These are not independent, but show strong regularities. Looking at a short excerpt, we see that it consists of oscillations over time (Figure 3.1a). We may thus be able to describe it more compactly in terms of the strengths of oscillations at different frequencies instead of the amplitudes over time. When using sinusoids as prototypical oscillations, such a description is called a *spectrum*, depicted in Figure 3.1b. It is obtained by multiplying the time-domain signal with all cosine and sine functions that have a whole number of oscillations over the excerpt – here, for a 40-sample excerpt, between 0 oscillations (a flat line) and 20 oscillations, shown in Figure 3.2 – and summing each of these products over time:

$$S_k = \frac{1 + [k > 0]}{T} \sum_{t=0}^{T-1} s_t \left(\cos\left(\frac{t}{T} 2\pi k\right) - i \sin\left(\frac{t}{T} 2\pi k\right) \right), \quad k \in (0, 1, \ldots, T/2) \quad (3.1)$$

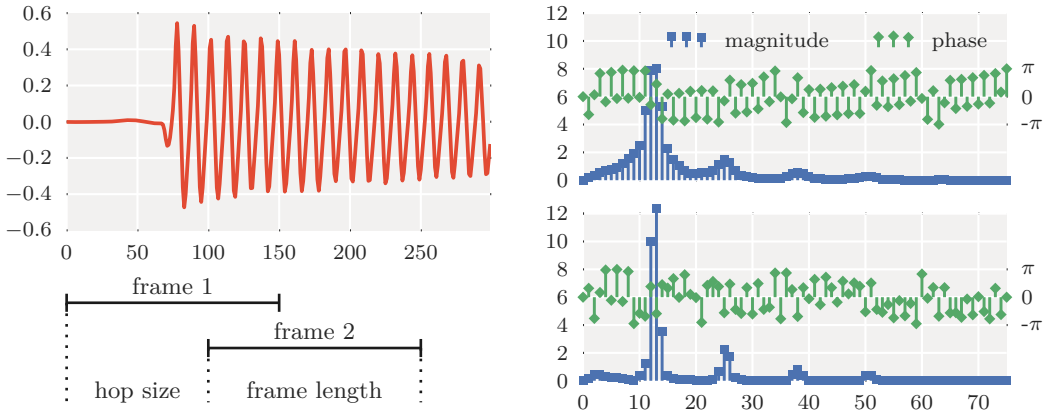
Each so-called *spectral bin* S_k is a complex number with the real part giving the unnormalized correlation with a cosine of k oscillations over the excerpt, and the complex part giving the correlation with a negative sine.[1] For the example in Figure 3.1b, we have $S_3 = -0.5 + i\sqrt{0.75}$, $S_{18} = 0.2$, and all other bins are zero. Due to the normalization factor $\frac{1}{T}$ (for $k = 0$) or $\frac{2}{T}$ (for $k > 0$), the bins can be directly interpreted as coefficients: The signal in Figure 3.1a is a sum of a cosine of 3 oscillations weighted by -0.5, a negative sine of 3 oscillations weighted by $\sqrt{0.75}$, and a negative sine of 18 oscillations weighted by 0.2.

Note that none of the other bins were active: Sinusoids of different whole-number oscillations are uncorrelated. Also note that S_{18} is fully imaginary: Cosines and sines of the same frequency are uncorrelated. Finally, note that the 42 template functions for the 40-sample excerpt can give at most 40 nonzero coefficients: S_1 to S_{19} can be complex, but S_0 and S_{20} are always real (because the corresponding sine functions are zero at every sample). Taken together, the set of template functions thus forms a complete orthogonal basis, so the transformation in Equation 3.1 is always invertible. It is known as the *Discrete Fourier Transform (DFT)*.

Often it is more useful to know the magnitude and temporal position for each oscillation, rather than the mixing coefficients of the cosine and sine reproducing it. This is accomplished by converting the complex bins from rectangular form $a + ib$ to polar form $m \exp(ip)$, where $m = \sqrt{a^2 + b^2}$ is the magnitude and $p = \arctan(b/a)$ is the phase. This form is depicted in Figure 3.1b (bottom). It shows that the signal is composed of a sinusoid of magnitude 1.0 and phase $2/3\,\pi$, and another of magnitude 0.2 and phase 0: $s_t = 1.0 \cos(t/40\, 6\,\pi + 2/3\,\pi) + 0.2 \cos(t/40\, 36\,\pi + 0)$.

The example signal behaves especially nicely, producing a very sparse spectrum. Real-world data often deviates from this. For a signal that contains non-sinusoidal oscillations, the spectrum has non-zero magnitudes both at the frequency of those oscillations (the *fundamental frequency*) and at multiples of this (the *harmonics*). Furthermore, if the signal excerpt contains sinusoids of other frequencies than the templates, i.e., sinusoids that do not fit a whole number of periods within the excerpt, they will correlate with *all* template functions – strongly with the closest ones in frequency, and weakly with others further away. More generally, this happens whenever the excerpt is not periodic, i.e., does not smoothly transition from the last sample to the first sample (as is the case for fractional frequencies). This can be mitigated by multiplying the signal excerpt with a *window function* that smoothly goes to zero near the beginning and end, before applying the DFT. While this also blurs the spectrum of signals composed of integer frequencies only, it produces similar spectral peaks for integer and fractional frequencies, facilitating further analysis. To follow this thesis, more detailed knowledge on the choice of window function is not required; see Lyons (2010, Sec. 3.8–3.9) if interested.

[1] The negative sign for the complex part is a common convention, not a necessity.



(a) signal sliced into frames

(b) corresponding spectra

Figure 3.3: A spectrogram is a sequence of spectra (b) computed from excerpts of a signal (a). The length and overlap of excerpts can be chosen freely.

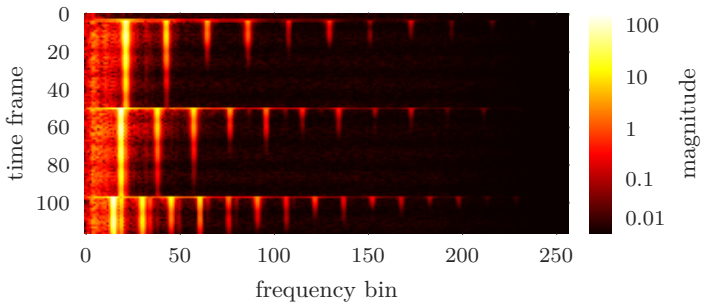
3.1.3 Spectrogram Computation

We have seen that the spectrum provides a very compact (or sparse) representation of a signal composed of oscillations of few frequencies that span the full signal. For long recordings, this is usually not fulfilled: Many oscillations will only be present over limited time spans, or change their frequency. The spectrum of such a recording still has peaks at the frequencies present in the signal, but nonzero magnitudes at all other bins as well, and no easy way to see when, how long or in what order different oscillations occur in the signal – this is determined by the phases, which coordinate all involved oscillations to create constructive and destructive interferences at the positions needed to represent the time-domain signal.

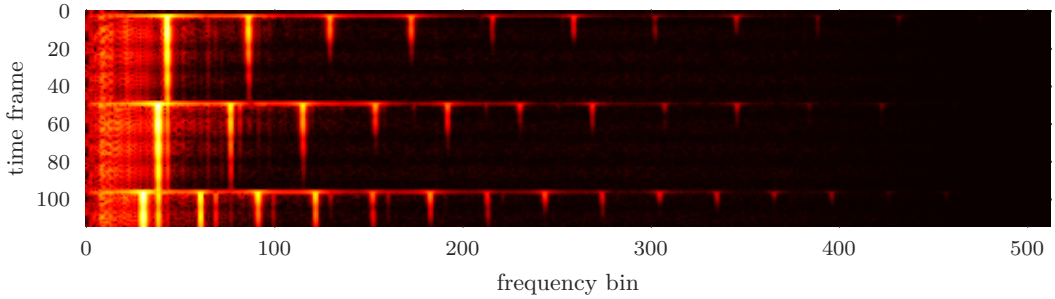
To obtain a representation that directly shows the timing and frequency of the signal constituents, we can slice the signal into small possibly overlapping excerpts, separately compute their spectra, and stack them in chronological order to form a matrix. The excerpts are referred to as *frames*, the operation as a *Short-Time Fourier Transform (STFT)* and the resulting matrix of spectra over time as a *spectrogram*. Formally, we have

$$X_{t,k} = \sum_{n=0}^{T-1} s_{Ht+n}\, w_n \left(\cos\Big(\frac{n}{T} 2\pi k\Big) - i \sin\Big(\frac{n}{T} 2\pi k\Big)\right), \quad k \in (0, 1, \ldots, T/2), \tag{3.2}$$

so that each time-frequency bin $X_{t,k}$ gives the magnitude and phase of a sinusoid of frequency k/T in the signal excerpt of T samples starting at position Ht. Figure 3.3 illustrates this operation for a short signal. Compared to Equation 3.1 on p. 47,



(a) frame length $T = 512$, hop size $H = 315$



(b) frame length $T = 1024$, hop size $H = 315$

Figure 3.4: Magnitude spectrograms of a 1.6-second electric guitar recording, with two different frame lengths. Shorter frames improve temporal resolution (vertical axis) at the cost of frequency resolution (horizontal axis).

we omitted the normalization factors,[2] and explicitly added multiplication of each signal frame with a window $\boldsymbol{w}$ (a vector of length T), as motivated on p. 48.

For a longer signal, the spectrogram can be visualized more compactly by mapping the magnitudes of time-frequency bins to colours of image pixels. Figure 3.4 demonstrates this for a recording of three electric guitar notes, with time progressing from top to bottom and frequency from left to right.[3] The onsets of the three notes are visible as horizontal lines, i.e., spectra with large magnitudes across most of the frequency range. These are the frames that overlap with the beginning of the oscillation of the guitar string (as in Figure 3.3b, top). The notes themselves show a comb-like pattern formed by a sequence of similar spectra that have large magnitudes mostly at the fundamental frequency and the harmonics.

[2] Most textbooks and implementations defer normalization to the inverse transformation. Besides, for the algorithms used in this thesis, input data is standardized so the scale does not matter.

[3] Spectrograms in this work will either follow this layout or have time from left to right and frequency from bottom to top, depending on what better suits the page layout.

The spectrogram computation has two freely-choosable parameters: The *frame length* T, and the *hop size* H. The hop size is the distance between the beginnings of two consecutive frames. When $H < T$, the frames overlap by $T - H$ samples. A common choice is $H = T/2$ (50% overlap): For many window functions, this is a good compromise between overlapping too little (causing samples that only fall into the tapered ends of windows to be weighted less than those falling into a centre) and wasting computation (for high overlap, spectra of consecutive frames will be strongly correlated and provide little information), as discussed in detail by Heinzel et al. (2002, Sec. 10). In this work, depending on the application, I often select H independently of T to obtain a target frame rate such as 100 frames per second.

The frame length T directly controls the number of spectral bins $T/2+1$. Doubling the frame length adds a bin between every pair of consecutive frequency bins, increasing the frequency resolution by a factor of 2. However, it also doubles the time support of each spectrum, encoding timing information within the frame in the phases. When looking at the magnitude spectrogram only, a larger frame length thus appears blurred in time, as demonstrated in Figure 3.4 for $T = 512$ and 1024 at the same $H = 315$ (70 frames per second at sample rate 22,050 Hz). As phases are often discarded, the frame length becomes an important tuning parameter: It needs to be chosen long enough to yield a suitable frequency resolution for a task, and short enough so timing information within the frames becomes irrelevant. For computational reasons, T is often furthermore constrained to a power of 2 – this allows to compute the DFT using the *Fast Fourier Transform (FFT)* algorithm by Cooley and Tukey (1965) in $O(T \log T)$ instead of the naive $O(T^2)$.

3.2 Perceptually-Informed Spectrograms

The spectrogram defined in Equation 3.2 has a linear time dimension, a linear frequency dimension and linear magnitudes (when decomposing the complex spectrogram into phase and magnitude). When trying to develop a system to replicate human capabilities of sound understanding, it may be useful to base it on what is known about basic sound perception. While human perception of time strongly depends on the context, scale and individual (Matthews and Meck, 2014), pitch and loudness perception is well-understood and can be approximately modelled by simple transformations of the spectrogram.

3.2.1 Frequency to Pitch

A spectrogram as computed with Equation 3.2 consists of spectra with a linear mapping from spectral bins to frequencies. Specifically, the k^{th} bin represents the magnitude and phase of a sinusoid of frequency k/T, where T is the frame length. For example, in Figure 3.4b, bins 42 and 84 represent 904.4 Hz and 1808.8 Hz,

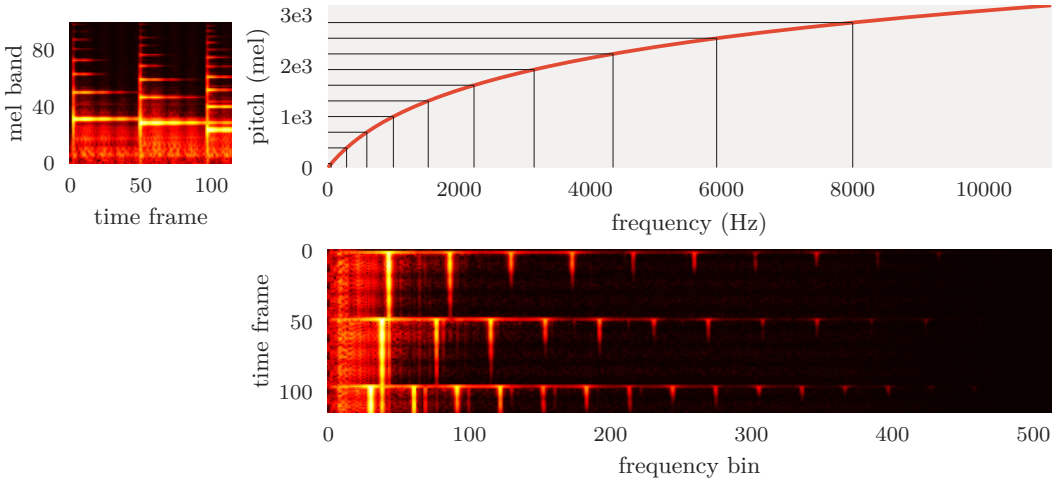


Figure 3.5: Illustration of the mel scale. The top right shows a plot of Equation 3.3 mapping the frequency range from 0 Hz to 11,025 Hz onto the mel scale, in which equal distances amount to equal perceived pitch differences. By warping the frequency dimension of a spectrogram (bottom right) from being proportional to Hz to being proportional to mel, we obtain a mel spectrogram (top left).

respectively. The human perception of pitch depends on the frequency, but it is not linear: a sinusoid of 1 kHz is not perceived half as high as a sinusoid of 2 kHz, and the distance between 1 kHz and 2 kHz is not perceived as the same as the distance between 2 kHz and 3 kHz or 1 kHz and 0 Hz.

Through an experiment asking 10 subjects to subdivide each of three given frequency ranges into four parts of equal pitch difference, Stevens and Volkmann (1940) obtained an approximate mapping of frequency to perceived pitch, which they verified with a second experiment asking 12 subjects to halve the pitch of eight given tones. Makhoul and Cosell (1976) later approximated this mapping further with a simple functional form:

$$m(f) = 1127 \log\left(1 + \frac{f}{700}\right) \tag{3.3}$$

The unit of pitch is referred to as a *mel*, using 1000 mel = 1000 Hz as a reference point, and its domain is termed the *mel scale*. Figure 3.5 (top right) visualizes the mapping from Hertz to mel. We see that evenly-spaced points on the mel scale correspond to points in frequency space that are spread nearly-logarithmically for high pitches ($f/700 \gg 1$), and nearly-linearly for low ones ($f/700 \ll 1$).

Note that this was neither the first attempt to quantify pitch perception (e.g., Preyer, 1876; Lorenz, 1890), nor the first attempt to approximate the mapping

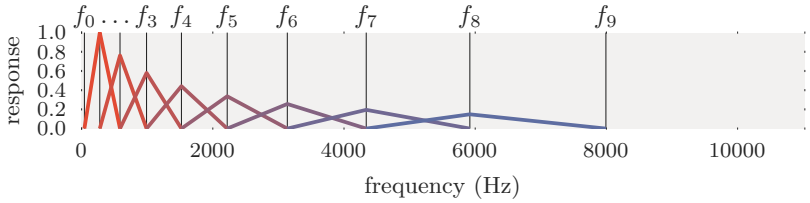


Figure 3.6: Mel filterbank with 8 overlapping bandpass filters

plotted by Stevens and Volkmann with a function (e.g., Koenig, 1949). Indeed, other functional forms like $m(f) = f/(af + b)$ provide a better fit to the plotted mapping (Umesh et al., 1999), and the whole idea of fitting functions to an interpolated plot rather than its few underlying data points seems suboptimal. However, the approximation in Equation 3.3 is used widely in literature and public software implementations, and also adopted for this work.

To mimic human pitch perception, we would like to transform the frequency axis of the spectrogram such that the distance between bins is proportional to the distance in mel rather than in Hz, as demonstrated in Figure 3.5 (top left). This can be accomplished with a filterbank, using a set of bandpass filters with centre frequencies spaced equally on the mel scale. The filterbank can be easily implemented by computing weighted sums of the frequency bins in a regular magnitude spectrogram. Specifically, for a mel spectrogram of M mel bands spanning the frequency range from $f_{\min}$ to $f_{\max}$, we map $f_{\min}$ and $f_{\max}$ to the mel scale using Equation 3.3, obtaining $m_{\min} = m(f_{\min})$ and $m_{\max} = m(f_{\max})$ as the lower bound of the first and the upper bound of the M^{th} bandpass filter, respectively. We then place M equally-spaced points on the mel scale between $m_{\min}$ and $m_{\max}$, serving as the centre pitches of the M filters:

$$m_i = m_{\min} + i\,\frac{m_{\max} - m_{\min}}{M+1} \qquad \text{for } i \in [1, M] \tag{3.4}$$

We map these points back to frequency space, obtaining $f_i = f(m_i)$ with

$$f(m) = 700\left(\exp\left(\frac{m}{1127}\right) - 1\right). \tag{3.5}$$

Finally, we construct M asymmetric triangular filters each with a peak at f_i and linear ramps towards zero at f_{i-1} and f_{i+1}, where $f_0 := f_{\min}$ and $f_{M+1} := f_{\max}$. This results in a bank of overlapping bandpass filters. Often, filters are normalized by their width, to obtain comparable magnitudes despite different band widths.

Figure 3.6 illustrates the construction of a mel filterbank of $M = 8$ bands from $f_{\min} = f_0 = 50$ to $f_{\max} = f_9 = 8000$ Hz. From 10 equally-spaced points on the mel scale we obtain the filter peaks and limits (shown with black lines in Figure 3.5) to create 8 overlapping triangular filters, normalized to the same summed response.

In practice, M is usually much larger than that, but also much smaller than the number of frequency bins in the original spectrogram – e.g., in Figure 3.5, the mel spectrogram has 100 bands, computed from a spectrogram of 513 bins. In addition to approximating human pitch perception, computing a mel spectrogram can thus also be seen as a lossy way of compressing a magnitude spectrogram, by decreasing frequency resolution for higher pitches.

There are several variations to this approach: Triangular filters can be replaced by rectangular filters or other forms, and applied in the time domain instead of post-processing a spectrogram. Instead of the mel scale, one can use the *Bark scale*, originally proposed as an approximation to *critical bands of hearing* (Zwicker, 1961), but later brought into a form similar to Equation 3.3 (Traunmüller, 1990). For some applications, it can also be useful to have a purely logarithmic frequency dimension, such that distances between bins indicate a fixed ratio of frequencies. In particular, a spectrogram can be transformed such that an octave – a frequency ratio of 2 – corresponds to a whole number B of bins. With $B = 12$, the ratio between consecutive bins amounts to a semitone on the equal-tempered Western musical scale, so it can be tuned to have the centre frequency of each bin match the frequency of a note in a Western music piece. This is sometimes referred to as a *cent-scaled* spectrogram, in reference to the *cent* (defined as the frequency ratio $2^{1/1200}$, such that 1200 cents correspond to an octave, and 100 cents to a semitone).

In this thesis, I limited myself to the mel scale. While cent-scaled spectrograms may seem ideal for processing music, they are based on assumptions that may not hold for non-Western music, and they require the tuning of a piece to be known or estimated. Furthermore, for the applications considered, the aspect of smartly compressing the spectrogram was more important than the exact choice of scale.

3.2.2 Magnitude to Loudness

The perceived loudness of a sound depends on the amplitude of the sound waves reaching the eardrum, which is captured by the magnitude in a spectrogram: Larger magnitudes correspond to a louder sound. Like pitch perception, loudness perception is not linear, so doubling the amplitudes (and magnitudes) does not result in a sound perceived twice as loud. And as for pitch perception, researchers have conducted empirical studies to quantitatively model loudness perception (of which I can only present a few selected examples here; see Marks and Florentine (2011) for a more complete review).

Fechner (1860) proposed to express perceived loudness l as the logarithm of magnitude m offset by the minimum perceptible magnitude m_0 (so its loudness is zero):

$$l(m) = \log(m) - \log(m_0) = \log(m/m_0) \tag{3.6}$$

Fechner postulated this as a general law for mapping stimulus intensities to sensation intensities, based on experiments by Weber (1846) suggesting that just-noticeable differences Δm of a stimulus intensity are proportional to the stimulus intensity m. Fechner's law is the basis for measuring sound pressure levels in *decibels (dB)* over the hearing threshold, where a decibel is the ratio $10^{1/10}$.

The logarithmic mapping was questioned in later studies. Richardson and Ross (1930) asked 11 subjects to assign numerical values to their perceived loudness of different test tones of known amplitudes, finding that loudness perception could be modelled better with a power function:

$$l(m) = m^a \tag{3.7}$$

Specifically, they reported an exponent $a = 0.44$ to provide a good fit to their data. After two decades of researching sound perception, Stevens (1955) also assumed a power function. Collecting data from 40 studies on loudness perception of sinusoids near 1 kHz, fitting a power function to each and taking the median of the best-fitting exponents, he arrived at $a = 0.6$, which he then verified in experiments of his own. With this exponent, an increase in magnitude by $\sqrt{10\text{ dB}}$ – or, equivalently, an increase in power (squared magnitude) by 10 dB – corresponds to a doubling of loudness (i.e., $l(\sqrt{10}m) \approx 2l(m)$). Stevens then defined the *sone* unit such that 1 sone corresponds to the loudness of a sinusoid of 1 kHz at 40 dB power over the hearing threshold. This unit is standardized in ISO/R 131:1959. Later, Stevens (1957) proposed the use of power functions as a general psychophysical law to replace Fechner's, backed by studies for different sense modalities.

However, neither (3.6) nor (3.7) are sufficient to capture human loudness perception. In contrast to pitch perception, which can be modelled as a one-dimensional mapping from frequency to pitch, it was quickly discovered that perceived loudness depends on at least two dimensions: magnitude and frequency. Fletcher and Munson (1933) published one of the first studies determining *equal-loudness curves*: Asking 11 subjects to adjust the amplitude of a sinusoid of 1 kHz to match the perceived loudness of a second sinusoid, over different frequencies and amplitudes, they quantified how loudness perception varies with frequency, at different loudness levels. For example, they found that a sinusoid of 1 kHz sounds louder than a sinusoid of the same amplitude at 100 Hz or 10 kHz, but softer than a same-amplitude sinusoid at 3 kHz. Their results were refined over time; an overview over succeeding studies is given by Suzuki et al. (2003) along with an improved approximation of equal-loudness curves standardized as ISO 226:2003. In deriving their curves from data of 19 studies, Suzuki et al. also found that perceived loudness can be fit best to a power function of the form

$$l(m, f) = b(f)\left(m^{a(f)} - m_0(f)^{a(f)}\right), \tag{3.8}$$

where m_0, exponent a and scale factor b all depend on the frequency f.

Note that all results presented above specifically focus on sinusoids. While complex tones can be expressed as a linear combination of sinusoids of different frequencies – as the spectrum does – the loudness of a complex tone is generally not just a linear combination of the loudnesses of the sinusoids. For example, for two sinusoids of similar frequency, the louder one will *mask* the softer one, up to rendering it imperceptible (Fletcher and Munson, 1933; Fletcher, 1940; Zwicker, 1961), and loudness perception for white noise deviates from tones (Stevens, 1957, p. 167).

With all this in mind, note that our purpose is not to estimate the perceived overall loudness of a recording, but to preprocess a spectrogram for further computational analysis. This has two implications:

(1) There is no need for and no benefit to expect from simulating interactions between different frequencies such as masking effects; we want to be able to analyse the components of a complex sound independently of the presence of other components.

(2) Before the spectrogram reaches a machine learning algorithm, it will be subject to further normalization such as standardization of the mean and variance per frequency band, cancelling the effects of simple pitch-dependent magnitude transformations.

Furthermore, we usually only know that the amplitudes of a digital recording are proportional to the amplitudes of the physical sound wave that was recorded, without any information on the scale. This furthermore implies that

(3) ideally, the combination of magnitude transformation and normalization should be scale-invariant, i.e., the resulting normalized spectrogram should not change when linearly scaling the input.

In practice, a very common approach – and the one I adopted in this thesis – is to scale magnitudes logarithmically, without any frequency dependency:

$$l(m) = \log(m) \tag{3.9}$$

This turns scaling of magnitudes into an offset ($l(cm) = \log(m) + \log(c)$) and exponentiation into a factor ($l(m^c) = c\log(m)$), both of which is cancelled when subsequently standardizing the spectrogram to zero mean and unit variance. Thus, it fulfils point (3) above, and when standardizing per frequency band, it also eliminates the need for a frequency-dependent scale or exponent as in Equation 3.8. A problem is that the logarithm is unbounded for small arguments: When a spectrogram contains near-zero magnitudes, they are transformed into very large negative values, and absolute silence is transformed to $-\infty$, both of which can be problematic for machine learning algorithms. A simple solution is to clip low magnitudes:

$$l(m) = \log(\max(a, m)) - \log(a) \tag{3.10}$$

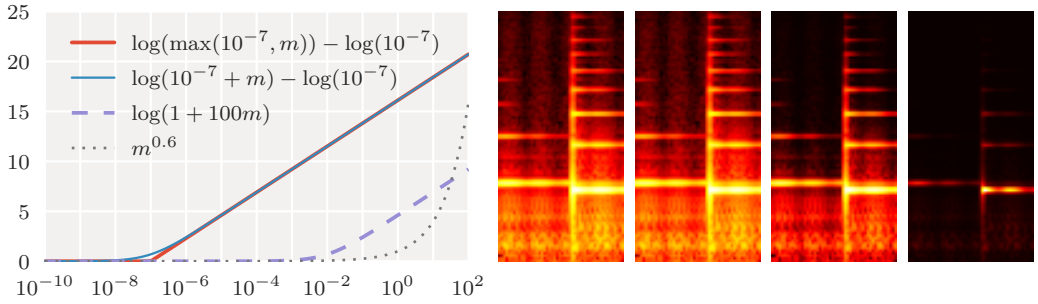


Figure 3.7: Four different magnitude scalings as a line graph (note the logarithmic horizontal axis) and applied to a mel spectrogram excerpt (same order as in the line graph's legend).

The offset $\log(a)$ ensures $l(m) \geq 0$, which is irrelevant when standardizing the spectrogram, but will prove convenient below for comparing alternative formulations. a is chosen small enough to not clip important information, but large enough to limit the output to a manageable range – in my work, I typically set $a = 10^{-7}$. Another solution is to add a small constant:

$$l(m) = \log(a + m) - \log(a) \tag{3.11}$$

$$= \log\Big(1 + \frac{1}{a}m\Big) \tag{3.12}$$

This is equivalent to adding a small amount of white noise to the spectrogram before taking the logarithm of magnitudes (since ideal white noise contains all frequency components in equal amounts). For large magnitudes, (3.10) and (3.11) behave the same, but towards zero, the latter is more linear, as shown in Figure 3.7. Some authors using the equivalent formulation (3.12) treat $1/a$ as a *compression factor* and deliberately set a to a much larger value such as 10^{-2} (Klapuri et al., 2006, Eq. 1). While not directly justifiable from a perspective of simulating human perception, this allows to limit the emphasis of low-magnitude structure in a spectrogram, and can work well depending on the application – besides, it is useful for visualization (as shown in Figure 3.7, a colour mapping based on $\log(1 + 100m)$ highlights the spectral peaks visible in Figure 3.3b without emphasizing low-magnitude noise in high frequencies). Unlike Equation 3.9, Equations 3.10–3.12 depend on the input scale. This is not pronounced for $a = 10^{-7}$, but for $a = 10^{-2}$, it makes a difference whether the spectrogram was normalized by the frame length (cf. Section 3.1.2). Finally, some authors use a power function (Eq. 3.7) such as $l(m) = m^{0.5} = \sqrt{m}$ (Camacho and Harris, 2007; Stark and Plumbley, 2009). Like logarithmic scaling, this ensures scale invariance after standardization (scaling the input scales the output), and emphasizes low magnitudes (for $a < 1$). Figure 3.7 visually compares the different options, the first of which is used for most of the work in this thesis.

3.3 Framewise Audio Features

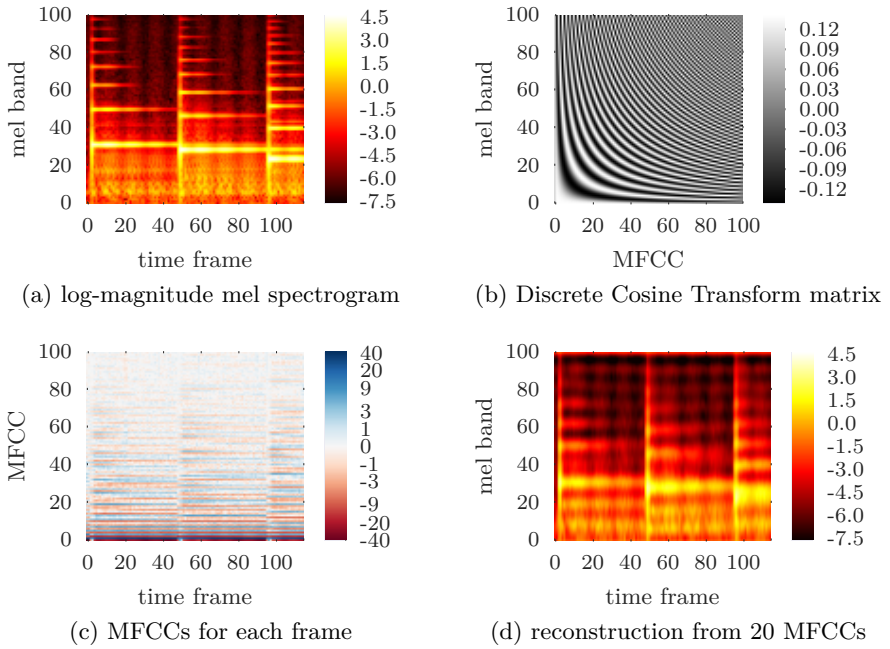
Several audio descriptors have been derived from the spectrogram and mel spectrogram. The most basic ones are computed from single spectrogram frames, i.e., single short-time spectra. We will briefly discuss a few that either help designing network architectures building on spectrograms, or help understanding prior work approaching the applications addressed in this thesis.

Mel-Frequency Cepstral Coefficients (MFCCs) are computed by applying a *Discrete Cosine Transform (DCT)* to the frames of a mel spectrogram with logarithmic magnitudes, often limited to the first few components. MFCCs have originally been proposed for analysing time series data for echoes (Bogert et al., 1963), then shown to work well for speech recognition (Davis and Mermelstein, 1980), and finally adopted for music processing (Foote, 1997). The DCT is very similar to the DFT discussed in Section 3.1.2, but instead of cosine and sine templates, it only uses cosine templates, and includes frequencies of $k + 0.5$ cycles over the input (for whole numbers k) to still form a complete basis. When applied to logarithmic spectral magnitudes, it provides a description of the spectrum in terms of periodicities over the frequency bins. Omitting the higher-order components can be seen as keeping a coarse, smoothed approximation of the spectral shape. It can also be seen as a means of decorrelating and compressing the spectrum: Both for speech (Pols, 1977, Tab. 2.3.2) and for music (Logan, 2000, Fig. 6), the spectra's first *principal components* appear similar to the DCT's cosine templates, so the DCT can be regarded as an approximation to a *Principal Component Analysis (PCA)* (although for music, this only holds superficially, as I detailed in Schlüter, 2011, Sec. 5.3). Figure 3.8 illustrates this computation for a mel spectrogram excerpt.

Spectral centroid, spread, skewness, kurtosis, slope, rolloff are simple scalar features describing the shape of a magnitude spectrum (i.e., a spectrogram frame). They are based on treating the spectrum as a probability distribution over frequency bins, by normalizing it to a sum of 1.0:

$$p(k) = \frac{S_k}{\sum_k S_k} \tag{3.13}$$

The centroid, spread, skewness and kurtosis are then defined as the mean, variance, and the normalized third and fourth-order central moments of this distribution, respectively. The slope is the coefficient a of a linear regression $ak+b$ fit to the spectrum S_k. Finally, the rolloff is defined as the 95-percentile of $p(k)$. Peeters (2004, Sec. 6.1) gives more complete definitions along with illustrations. All these features capture basic aspects of the spectral shape, al-



(a) log-magnitude mel spectrogram

(b) Discrete Cosine Transform matrix

(c) MFCCs for each frame

(d) reconstruction from 20 MFCCs

Figure 3.8: Multiplying each frame of a log-magnitude mel spectrogram (a) by a DCT matrix (b), we obtain Mel-Frequency Cepstral Coefficients (c). Most energy is contained in the lower MFCCs, which capture a coarse approximation of the spectra (d).

lowing to distinguish frames of dominantly low and high frequencies (centroid, skewness and slope), or noisy and harmonic frames (kurtosis).

Chroma vectors describe a spectrum in terms of the total magnitudes assigned to the twelve semitones of the Western equal-tempered scale, collapsing all octaves into one. In their simplest form, introduced by Fujishima (1999) and Wakefield (1999), this just requires computing twelve different weighted sums of the frequency bins S_k, each collecting the magnitudes belonging to a particular semitone over all octaves. More complex implementations include contributions of harmonics (Gómez, 2006, Sec. 3.4), use phase information (in particular, instantaneous frequency) for more accurate low-frequency pitch estimates (Ellis and Poliner, 2007) or estimate the tuning of the piece for better alignment with the notes (Zhu et al., 2005).

Delta and acceleration features can be computed from any frame-based audio feature by computing the difference between consecutive frames (delta), or the difference between consecutive deltas (delta-delta, or acceleration). They allow a framewise classifier to assess short-term temporal evolution of features.

3.4 Blockwise Audio Features

For many tasks, the information in a single frame (or three frames, if including delta and acceleration features) is not enough to make a prediction. Music is a process over time and thus music understanding tasks commonly require evaluating a nontrivial amount of temporal context. For example, estimating the tempo of a music piece requires to detect at least two beats, and for distinguishing instruments playing the same note, the temporal loudness envelope is as important as (Heng et al., 2011) or even more important than (Prentiss et al., 2016) the spectrum. On the other hand, even seemingly complex tasks can be solved with just a few seconds of context: In a study by Gjerdingen and Perrott (2008, Sec. 3.2), subjects were able to identify ten music genres from 3-second excerpts just as well as from longer excerpts (agreeing with the distributor's genre labels about 70% of the time). For different applications, researchers have thus designed features that describe up to a couple of seconds of an audio signal, often derived from framewise features or spectrogram excerpts. Again, we will briefly describe a few well-chosen examples.

Feature statistics are a simple meta-feature derived from any framewise feature by computing its mean, variance, median, minimum, maximum or any other quantile over a short excerpt. While this does not capture the temporal evolution of a feature, statistics over an increased temporal context can be more stable than framewise feature values, or more discriminative for a task (Scheirer and Slaney, 1997; Tzanetakis and Cook, 2002).

Fluctuation patterns are designed to describe rhythmic qualities of an excerpt. Proposed by Pampalk et al. (2002), the idea is to compute a magnitude spectrum of each frequency band of a log-magnitude spectrogram excerpt. This captures temporal periodicities within the excerpt, for different frequencies.[4] Keeping only low periodicities, e.g., between 0 and 10 Hz, fluctuation patterns can be seen as a rhythm descriptor. For music similarity estimation, Pampalk (2006, pp. 36–42) computes fluctuation patterns from a 3-second mel spectrogram excerpt with reduced frequency resolution (e.g., to 12 bands), dampens the lowest periodicities that carry most energy, emphasizes differences between neighbouring periodicities, and blurs the result to facilitate comparisons of two patterns. Other variants map periodicities to a logarithmic scale (Jensen et al., 2009; Seyerlehner et al., 2010b), such that a tempo change results in a translation along the periodicity axis, or preprocess the spectrogram excerpt with a simple onset filter that computes the temporal difference followed by linear rectification (Jensen et al., 2009; Pohle et al., 2009).

[4] Note the duality to MFCCs, which capture periodicities over frequencies for each frame. Using magnitude spectra instead of the DCT makes the features phase invariant, reducing dependency on the temporal position of the excerpt in the music piece.

Block-level Features (BLFs) are a range of features computed from spectrogram excerpts, hand-designed for music similarity estimation by Seyerlehner et al. (2010b), then used for tag prediction and genre classification (Seyerlehner et al., 2010a). They serve here as an example of elaborated feature design, but will also play a central role in Chapter 6. Seyerlehner's *Spectral Pattern (SP)* is computed from a 10-frame cent-scaled log-magnitude spectrogram excerpt (with ~43 frames/sec) by sorting the magnitudes within each frequency band by value. The *Delta Spectral Pattern (DSP)* computes the rectified temporal difference between each frame and the frame two frames earlier[5] in a 25-frame excerpt, then sorts each frequency band by value. The *Variance Delta Spectral Pattern (VDSP)* computes the difference between frequency bands spaced two bands apart[6], then sorts each band by value. The *Logarithmic Fluctuation Pattern (LFP)* is one of the Fluctuation Pattern variants described before. The *Correlation Pattern (CP)* computes the correlation matrix of frequency bands in a 256-frame excerpt reduced to 52 bands. Finally, the *Spectral Contrast Pattern (SCP)* computes the *spectral contrast*, the difference between the maximum and minimum magnitude in a subband of a few frequencies, for 20 subbands in a 40-frame excerpt, then sorts each band by value.

To obtain a fixed-length descriptor of a song, all six patterns are computed for overlapping excerpts (with a custom hop size for each pattern) and then aggregated by taking their variance (for the VDSP) or custom quantiles (for all others).

While the features may seem ad hoc, they have been carefully chosen and tuned to perform well on the *1517-artists* genre classification dataset collected by Seyerlehner, and produced state-of-the-art results in similarity estimation and classification across multiple datasets.

Having learned about or refreshed our mind on audio signal processing and feature extraction, we are now ready to combine it with deep learning.

[5] Seyerlehner et al. (2010b, Sec. 3.2.2) said three frames, but their code used two.

[6] Seyerlehner et al. (2010b, Sec. 3.2.3) said to compute the temporal difference, but their code computed the difference over frequencies.

4 Connecting Audio Signal Processing and Deep Learning

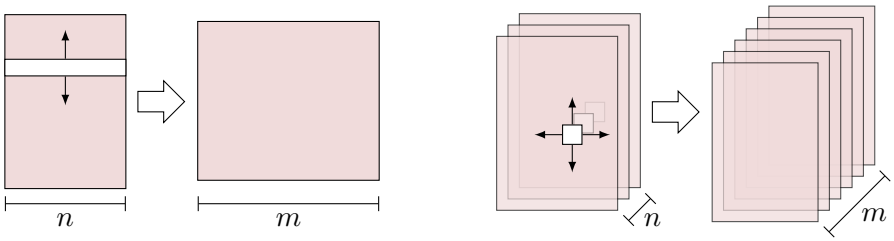
Before we dive into the main part of this thesis, let us draw some connections between what we just learned about audio and music signal processing to what we learned about neural network architectures in Chapter 2, and, in a second meaning of the chapter title, reason about ways to connect the output of a signal processing algorithm to a neural network.

4.1 Signal Processing Algorithms as Neural Networks

In the following three subsections, we will compare the basic audio signal processing algorithms discussed in Chapter 3 to network layers of Chapter 2 and point out how the computational steps coincide.

4.1.1 Spectrogram computation as 1D convolution

As we noted in Section 3.1.2 (p. 47), the spectrum of a signal excerpt is computed by taking the elementwise product with different template functions and summing each product over time, i.e., by computing the dot product between the excerpt and each template function. An optional multiplication of the excerpt by a window function can be absorbed into the templates.



(a) 1D convolution (b) 2D convolution

Figure 4.1: Comparison of the 1D (a) and 2D (b) convolution operations as used in CNNs. Both compute m outputs from n inputs, retaining the spatial layout along the convolution dimension(s) and discarding it along the feature dimension. The filter tensor (white) for each output matches the size of the input tensor (tinted) except in the convolution dimension(s), for which the size can be chosen freely.

To obtain a spectrogram of a signal, we concatenate the spectra of multiple same-length signal excerpts with starting points spaced H samples apart (Section 3.1.3, p. 49). For $H = 1$, each frequency band $\boldsymbol{X}_{:,k}$ of the resulting spectrogram $\boldsymbol{X}$ is thus a one-dimensional convolution of the signal vector $\boldsymbol{s}$ with an appropriately defined complex template vector $\boldsymbol{C}_k$:

$$C_{k,n} := w_n \left(\cos\Big(\frac{n}{T} 2\pi k\Big) - i \sin\Big(\frac{n}{T} 2\pi k\Big) \right) \tag{4.1}$$

$$X_{t,k} = \sum_{n=0}^{T-1} s_{t+n}\, w_n \left(\cos\Big(\frac{n}{T} 2\pi k\Big) - i \sin\Big(\frac{n}{T} 2\pi k\Big) \right) = \sum_{n=0}^{T-1} s_{t+n}\, C_{k,n} \;\Leftrightarrow\; \boldsymbol{X}_{:,k} = \boldsymbol{s} * \boldsymbol{C}_k \tag{4.2}$$

Recalling Section 2.2.3 (p. 18), a convolutional unit in a neural network performs a convolution of each of n input tensors (e.g., time series, or images) with a same-dimensional, but smaller weight tensor – a different one for each input tensor – and sums up the results to produce a single output tensor. A convolutional layer consists of m such units with different weight tensors, producing m output tensors from its n input tensors. Figure 4.1 illustrates this operation for 1D and 2D convolution.

In the case of a monophonic audio signal, the input is a single time series, so a convolutional unit would perform a single 1D convolution. With an appropriately chosen weight vector, it can thus produce a frequency band of a spectrogram, and a 1D convolutional layer can produce a full spectrogram (Figure 4.1a with $n = 1$). In practice, $H \gg 1$, but this can be implemented as a *strided convolution* that only evaluates every H^{th} step. Furthermore, weights are usually not complex, but as the input is real, computation can be split, using twice as many units.

4.1.2 Framewise feature computation as 1D convolution

In Section 3.3, we looked at features computed from single spectrogram frames. If the framewise computation is a linear transformation, this can be expressed as a 1D convolutional layer: With a filter width of 1, a 1D convolution degrades to a scaling operation, so when regarding the spectrogram as an array of n time series (one per frequency band), a 1D convolutional unit computes a weighted sum of the n inputs per time step, and m such units compute a linear transformation $\mathbb{R}^n \to \mathbb{R}^m$ of each spectrogram frame (cf. Figure 4.1a). Thus, a single 1D convolutional layer could replicate mel-scaling, the DCT in computing MFCCs, or basic Chroma computation. More complex features could be computed from multiple 1D convolutions with nonlinearities in between. Delta and acceleration features are weighted sums of two or three consecutive frames, and thus replicable with a 1D convolutional layer of a filter width of 2 or 3.

4.1.3 Blockwise feature computation as 1D convolution

Blockwise features as discussed in Section 3.4 are computed from spectrogram excerpts. Again, if this computation is a linear transformation of the excerpt, it can be expressed as a 1D convolutional layer: With a filter width of l, a 1D convolutional layer applies a linear transformation $\mathbb{R}^{ln} \to \mathbb{R}^m$ to vectorized l-frame excerpts, with consecutive excerpts overlapping by $l-1$ frames (or $l-H$ frames for a strided convolution of hop size H). A single such layer could thus compute fluctuation patterns – despite their apparent complexity, they are based solely on linear operations (except for the optional onset filter, which would require an additional initial convolution and a linear rectifier). The other examples given in Section 3.4 include nonlinear operations and would thus either require a deep network of multiple convolutions interspersed with nonlinearities, or casting the operations as network layers (variance computation is differentiable, and sorting or taking a quantile are differentiable except at ties, which can be broken as described for the linear rectifier and max-pooling on p. 26).

4.2 Design Choices for Audio Processing with Deep Learning

When tasked with designing a neural network architecture for processing audio data, there are several choices to be made: How should we preprocess the audio signal? Should we precompute a spectrogram, apply frequency and/or magnitude scaling, extract hand-designed audio features, or rely on the network to learn everything or parts of this from the training data? What kind of network layers should we use? In the following, we will settle some of these questions for the remaining chapters.

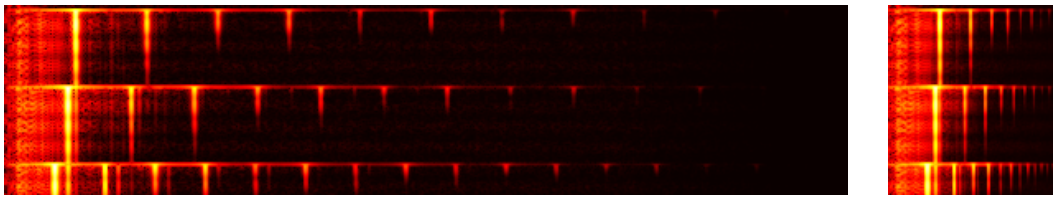
4.2.1 Waveforms vs. spectrograms

As we noted in the previous sections, a single 1D convolutional layer could learn to compute a spectrogram, and further 1D convolutional layers could replicate many hand-designed audio features based on spectrograms. A potential network architecture for audio data would thus consist of 1D convolutional layers processing the time-domain signal. Dieleman and Schrauwen (2014) indeed attempted this for a music tag prediction task, but even when helping the network to learn phase-invariant features (by summing the squared activations of pairs of convolutional units, mimicking the computation of magnitudes from complex spectra), it was inferior to starting from a magnitude spectrogram instead. Similarly, Hertel et al. (2016) found CNNs on spectrograms to outperform CNNs on time-domain signals for acoustic event detection in indoor and outdoor field recordings. Although for speech recognition, networks trained on raw audio signals have recently been catching up (Sainath et al., 2015; Ghahremani et al., 2016; van den Oord et al., 2016a; Sailor and Patil, 2016), this thesis will be limited to learning from spectrograms. Learning from the raw signal did not seem promising enough to warrant the additional computational complexity.

4.2.2 1D vs. 2D convolution of spectrograms

Settling on precomputing spectrograms, we could still use an architecture of only 1D convolutional layers, motivated by the fact that this restricted search space includes many well-working hand-designed features. Now note that in general, a 1D convolutional layer only retains the temporal layout of the spectrogram: It is oblivious to the order of the n frequency bands in exactly the same way a dense layer is oblivious to the order of its inputs (Section 2.2.3, p. 18), and also the ordering of its m output features is arbitrary (Figure 4.1a). This seems potentially wasteful. Mel filtering or chroma computation – both linear transformations replicable by a 1D convolutional layer – retain a spatial relationship of output features due to their sparse connectivity: Mel bands are weighted sums of neighbouring frequency bands, and chroma features are weighted sums of frequency bands spaced one or more full octaves apart, with nearby mel bands or chroma features computed from nearby frequency bands. As a special case of sparse connectivity, 2D convolutional layers also retain the frequential layout of a spectrogram by computing outputs from neighbouring bands, with the additional constraint that groups of outputs use the same filters applied at different frequency offsets. In this way, a 2D convolutional layer can produce m 2D feature maps from a spectrogram (Figure 4.1b for $n = 1$).

When designing a neural network to process spectrograms, there are arguments both for 1D and 2D convolution. 1D convolution acknowledges the fact that unlike in images, the two dimensions in spectrograms have a very different meaning.



(a) linear frequency scale (b) mel scale

Figure 4.2: Linear-frequency spectrogram (a) and mel-scaled spectrogram (b) of three electric guitar notes, with time progressing from top to bottom and frequency or pitch increasing from left to right. The mel spectrogram uses a nearly-logarithmic frequency transformation, such that harmonics form a pitch-independent pattern, while in the linear-frequency spectrogram they spread out wider for higher pitches.

Convolving only over time learns features that take into account the full frequency range, but can occur at any time in an input signal. 2D convolution assumes that features in spectrograms are not only localized and shift-invariant in time, but also in frequency or pitch, shiftable across the full range. To a certain degree, this is valid, as shown in Figure 4.2: For linear-frequency spectrograms, harmonics of a complex tone at a particular pitch are spaced with the same distance across the frequency range, and for mel-scaled spectrograms, transposing a tone along with its harmonics corresponds to translation.[1] Furthermore, on a more local level, spectrograms exhibit small-scale structure such as oriented edges practically at any location. Note that 1D and 2D convolutional layers are not mutually exclusive: We can design a network to have 2D convolutions for local features followed by 1D convolutions for features spanning all frequencies (but not vice versa, since 1D convolution discards the order of frequency bands).

The optimal choice can probably only be determined empirically, and may also depend on the task. In literature, both forms can be found. For speech recognition, early work relied on fully-connected layers only (Mohamed et al., 2009), with 1D convolution (Lee et al., 2009b) and 2D convolution (Abdel-Hamid et al., 2012) following later (although pilot studies using 1D convolution on a small subtask have been performed much earlier, e.g., Waibel et al., 1989). Similarly, for music analysis, early adopters of deep learning used fully-connected networks (Hamel et al., 2009; Nam et al., 2011), followed by 1D convolution (Lee et al., 2009b; Li et al., 2010; Dieleman et al., 2011) and 2D convolution (Humphrey and Bello, 2012; Schlüter and Böck, 2013). Recent work indicates that 2D convolutions provide an

[1] Musical transposition corresponds to translation on a logarithmic frequency scale. The mel scale is only approximately logarithmic (p. 52), so the correspondence only holds approximately.

advantage over using 1D convolutions only, both for speech recognition (Amodei et al., 2015, Sec. 3.5) and music analysis (Lostanlen and Cella, 2016, Tab. 2).

For the work in this thesis, I started with fully-connected networks (Chapter 5), then switched to 2D convolution (Chapters 7 and 8) and 2D convolution followed by 1D convolution (Chapter 9).

4.2.3 Linear vs. mel-scaled frequencies

Most features described in Sections 3.3 and 3.4 build on a mel-scaled spectrogram, motivated by its presumed simulation of human pitch perception (Section 3.2.1). When using a neural network instead of hand-designed features, we could alternatively start from a linear-frequency spectrogram and rely on the network to find a suitable transformation for the task at hands.

When processing a spectrogram with a 1D convolutional layer, the mel-scaling step may indeed be unneeded: A 1D convolutional unit jointly processes all frequency bands, so it could simply learn the linear mel-scaling operation by itself.[2] However, a hard-coded mel scaling step constrains the solution space, reducing frequency resolution in a particular, non-uniform way. This could be good (preventing the network from overfitting to unneeded details), bad (blurring details that would have been important) or effectless. In any case, it reduces computational complexity by reducing the size of the input to be processed by the network.

For a 2D convolutional layer, mel-scaling makes a more significant difference. Since it learns features that are local and shared between frequency bands, warping the frequency dimension affects what kind of operations it can perform. For example, a linear frequency scale makes it easy to detect different pitches, using filters that match a particular spacing of harmonics anywhere in the spectrum, whereas mel-scaled frequencies facilitate distinguishing the harmonic spectra of different instruments, with filters shared across a range of pitches. Only when constraining 2D convolutional processing to detecting very local features such as edges – e.g., using a single layer with 3×3 filters – mel-scaling will not make a difference besides reducing computational costs.

Again, the optimal choice is probably task-dependent and will have to be determined empirically, since there is no clear theoretically-justified advantage of one over the other. In literature, most researchers dedicated to deep-learning-based music analysis concurrently to me used either mel-scaled or logarithmic-frequency spectro-

[2] The attentive reader may wonder about the logarithmic magnitude scaling usually performed after mel-scaling (e.g., Pampalk, 2006; Pohle, 2010; Seyerlehner, 2010). This could be learned as well (Section 4.2.4), built into the network, or applied before mel-scaling: While not the same, early implementations also applied magnitude scaling first (Foote, 1997; Logan, 2000), and Molau et al. (2001) showed that at least for speech recognition, the order does not matter in practice.

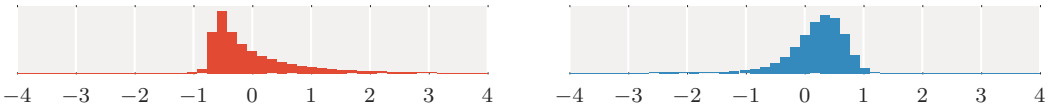
grams: Hamel (2012) for tagging, Humphrey (2015) for timbre similarity and chord estimation, Böck (2016) for onset detection, beat tracking, tempo estimation and piano transcription, Dieleman (2016) for tag prediction, metric learning and genre recognition, Raffel (2016) for metric learning, Sigtia (2016) for piano note and chord transcription. Hamel and Eck (2010) used linear spectra with a non-convolutional network architecture, but noted that this lead to long training times, and moved to PCA-compressed mel spectra in later work (Hamel et al., 2011), without giving a direct comparison. Much similarly, Sigtia and Dixon (2014); Sigtia et al. (2015) used linear spectra with a non-convolutional architecture and subsequently switched to logarithmic-frequency spectra without directly comparing the two. Boulanger-Lewandowski (2014) used PCA-compressed linear spectra for piano transcription and chord recognition, noting in passing that they found mel scaling to be unnecessary due to the subsequent compression (Boulanger-Lewandowski, 2014, p. 88).

For the work in this thesis, I mainly used mel-scaled spectrograms, both due to the prevalence in prior work and to benefit from reduced training times, but compare results to linear-frequency spectrograms in Chapter 9.

4.2.4 Linear vs. logarithmic magnitudes

Many of the features described in Sections 3.3 and 3.4 are based on logarithmic-magnitude spectrograms, both because logarithmic magnitudes are closer to human loudness perception than linear magnitudes (Section 3.2.2), and because they work better in practice. When training a neural network on spectrograms, we could start with linear magnitudes instead and assume the network will find an optimal transformation.

As proven by Arora et al. (2016, Sec. 3.2), a neural network of a hidden layer of m units with the linear rectifier nonlinearity (Equation 2.13, p. 16) and a linear output unit can express any piecewise linear function $\mathbb{R} \to \mathbb{R}$ with m pieces. Thus, in theory, the first two layers of a network processing spectrograms could learn a piecewise linear approximation to a logarithmic function, or any other nonlinear transformation of magnitudes. However, with most network architectures, the initial layers process patches of multiple spectral bins at once, so it makes a crucial difference whether input magnitudes are linear or logarithmic. To force the network to separately transform the magnitudes first, we would need to use two *pointwise convolutions* (2D convolutional layers with 1×1 filters) of m units and 1 unit, respectively – this is equivalent to processing each spectrogram bin with a fully-connected mini-network of m hidden units and 1 output unit (Lin et al., 2014), and has also been successfully applied to images to learn a colour space transformation (Mishkin et al., 2016, Tab. 6). But even then, it is unclear whether linear magnitudes would provide a better starting point than logarithmic magnitudes. After all, in contrast to mel scaling, scaling the magnitudes does not discard any information.



(a) standardized linear magnitudes (b) standardized logarithmic magnitudes

Figure 4.3: Linear mel spectral magnitudes (computed for all songs of the Jamendo dataset (Ramona et al., 2008), standardized to zero mean and unit variance per frequency band, binned together) roughly follow an exponential distribution, logarithmic magnitudes are more Gaussian.

So again, to reach a decision, let us consult the literature. Among contemporary researchers on deep learning for music analysis, there is a clear prevalence of logarithmic (Hamel, 2012; Humphrey, 2015; Böck, 2016; Dieleman, 2016; Raffel, 2016; Sigtia, 2016) or cubic root (Boulanger-Lewandowski, 2014) compressed magnitudes. Only Schmidt (2012) trained RBMs on linear-magnitude spectra. Also for speech recognition, most neural-network-based approaches on spectrograms use logarithmic magnitudes (e.g., Waibel et al., 1989; Dahl et al., 2010; Mohamed et al., 2012; Abdel-Hamid et al., 2012; Sainath et al., 2013c; Sercu et al., 2016), only few use linear magnitudes (e.g., Lee et al., 2009b; Amodei et al., 2015).[3] Unfortunately, there seems to be no direct empirical comparison of logarithmic and linear magnitudes for deep models in literature. For shallow models, Hsu et al. (2016, Sec. 4.3.4) obtain better genre recognition rates with logarithmic magnitudes, and Thickstun et al. (2017) observe improved performance with logarithmic magnitudes for polyphonic note detection. This could indicate that logarithmic magnitudes are suited better for classifiers, but it can be argued that shallow models could not have learned logarithmic transformations anyway. An interesting aspect is noted by Nam et al. (2012, Sec. 2.1.3): Linear spectral magnitudes tend to follow an exponential distribution, so taking the logarithm makes the distribution more Gaussian (see Figure 4.3). However, while some theoretical analyses of neural networks assume Gaussian-distributed input (e.g., Glorot and Bengio, 2010; Saxe et al., 2014; He et al., 2015), I am not aware of any work asserting that this would help learning.

For the work in this thesis, I used logarithmic magnitudes throughout. This was originally based on the fact that almost all prior work I consulted (both neural networks for speech, and hand-designed methods for music) used logarithmic magnitudes, and the idea of mimicking loudness perception seemed plausible enough. Only later I backed this with empirical evidence: While not reported in any of my publications, during experimentation I sometimes tried replacing logarithmic with linear magnitudes, always obtaining worse results.

[3] Mohamed et al. (2012) and Abdel-Hamid et al. (2012) do not mention using a logarithm, but in Hinton et al. (2012a, p. 88, top left), the same authors indicate they did.

5 Music and Speech Detection

In this chapter, we will look at the first sequence labelling task of this thesis: Detecting where in an audio recording there is music, where there is speech, and where there is both or neither.

When I worked on this topic in early 2012, existing solutions either relied on general-purpose audio features, or built on features specifically engineered for the task, combined with comparatively simple classifiers. I investigated whether an unsupervised feature learning method from computer vision – specifically, a mean-covariance Restricted Boltzmann Machine (mcRBM) – could be applied to spectrograms to learn competitive features automatically. The features it learned partly resemble engineered features, but outperform three hand-crafted feature sets in speech and music detection on a large corpus of radio recordings. This demonstrates that unsupervised learning can be a powerful alternative to knowledge engineering.

This work has previously been published in Schlüter and Sonnleitner (2012), in collaboration with Reinhard Sonnleitner, who helped coordinating the recording and annotation of the dataset, but was otherwise not involved in the experiments or in writing. This chapter mostly follows this publication, with some elaboration on related work, additional figures, and two new final sections.

The remainder of this chapter is organized as follows: Section 5.1 introduces the task and general idea in more detail, Section 5.2 reviews existing work on speech and music detection, and Section 5.3 gives an introduction to mcRBMs and their application to audio data. In Section 5.4, I apply the system to a large corpus of radio broadcasts, analyse the features learned by the mcRBM and evaluate its classification performance in comparison to three approaches using hand-crafted features. Section 5.5 briefly describes follow-up experiments done after the publication of Schlüter and Sonnleitner (2012), and Section 5.6 concludes with a discussion of the results and a reflection from today's perspective.

5.1 Introduction

Radio broadcasts are composed of two main types of audio content: Speech and music. Discriminating them is a basic first step in making their content accessible to further information retrieval. Often, music and speech do not just alternate, but overlap, and some applications require both classes of content to be detected independently. For example, in automatic broadcast transcription, only segments that contain speech should be passed to a speech recognition system. In this scenario, speech detection must be invariant to background music: Some radio stations constantly play music, even during the news, and a simple speech/music *discriminator* may give wrong predictions in this case. Another application is in collecting royalties from radio stations: In many countries, performance rights organizations charge royalties depending on the amount of music played, sometimes with a lower rate for music overlaid by speech. Automatic discrimination between pure music, music with speech, and non-music would facilitate a fair distribution of charges.

Existing approaches to speech and music detection either train standard classifiers on general-purpose audio features, or design new features based on observations on the structure of speech or music signals. A promising alternative is to learn features from data instead – in computer vision, learned features often outperform engineered features in object recognition tasks (Lee et al., 2009a). A particularly successful model for learning features from images, the mean-covariance Restricted Boltzmann Machine (mcRBM) (Ranzato and Hinton, 2010), has already successfully been applied to spectrograms of speech (Dahl et al., 2010) and music (Schlüter and Osendorfer, 2011), and thus seems to be an ideal candidate for our task. In this work, I employ a mcRBM to build a speech and a music detector.

5.2 Related Work

Many methods for detection or discrimination of speech and music have been proposed in literature. I will describe selected examples to highlight common strategies.

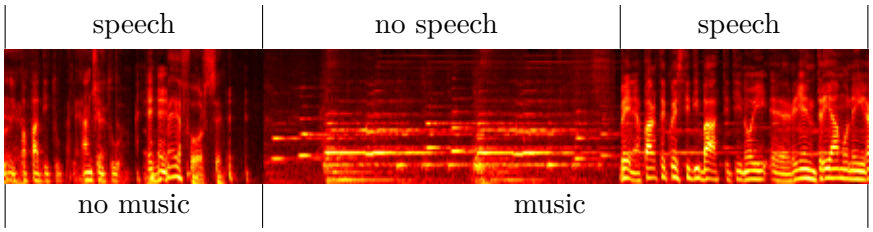


Figure 5.1: Mel spectrogram for a 10-second recording of pure speech, pure music, and speech with faint background music. The task addressed in this chapter is to detect speech and music independently of each other. This is in contrast to speech/music discrimination, which only distinguishes the first two cases (pure speech and pure music).

One class of approaches uses general-purpose audio features, hoping to capture distinct properties of speech or music. As speech is a sequence of vowels and consonants, it tends to exhibit quick changes in Zero Crossing Rate (ZCR), while music has a relatively constant ZCR. With a classifier trained on statistics capturing the ZCR variability, Saunders (Saunders, 1996) achieves a speech/music discrimination accuracy of 90%, and adding an energy contour feature improves accuracy to 98%. Carey et al. (1999) compare Mel-Frequency Cepstral Coefficients (MFCCs, see p. 58), amplitude, pitch and ZCR features classified with Gaussian Mixture Models (GMMs). They observe best performance in music/speech discrimination for MFCCs supplemented with their first-order derivative (i.e., delta). Pinquier et al. (2002) train a speech detector on MFCCs augmented with delta-MFCCs, and a music detector on spectral frames. Both detectors use GMMs for classification, and achieve an accuracy of 99.5% for speech and 93% for music detection, respectively. Liu et al. (2007) extract Linear Predictive Coefficients (LPCs), Line Spectral Pairs (LSPs), MFCCs, ZCR, and the spectral centroid, flux, rolloff, and kurtosis (see p. 58), augmented with the variance of each feature over a window of 1.28 s. Classification with a Multi-Layer Perceptron (MLP) yields an F-Score of 98% for music and speech detection.

Other authors design very smart features exploiting inherent characteristics of music or speech signals. A basic observation is that music contains long sustained tones of constant frequency (see Figure 5.1, central segment). Hawley (1993, pp. 78–87) implements an algorithm which finds frequency peaks in spectral frames, then calculates the average peak duration in a time window to detect the presence of music. Voiced speech, on the other hand, contains harmonics that quickly vary in frequency (see Figure 5.1, right segment). Hawley (1993, pp. 112–115) uses a comb filter to detect such harmonics, and filters for varying pitches to distinguish speech from music. Minami et al. (1998) detect sustained peaks as music, remove

them from the spectrogram and assume remaining harmonics to indicate speech. They report 90% and 80% accuracy for music and speech detection, respectively. Zhu et al. (2006) assume that most music is tuned to equal temperament, and design a feature assessing whether spectral peaks are tuned to a common reference pitch which does not change over time. They report about 96% precision and recall in music detection, even when mixed with speech. Seyerlehner et al. (2007) propose an improved detector for sustained peaks dubbed Continuous Frequency Activation (CFA), obtaining about 90% music detection accuracy on a corpus of TV broadcasts. Scheirer and Slaney (1997) design a set of features based on further observations about speech and music signals: the amount of low-energy frames per time unit (indicating natural pauses occurring in speech), the 4 Hz modulation energy (matching the syllabic rate of speech), a "pulse metric" (detecting strong steady rhythms), and the reconstruction error of a spectral frame from a smoothed cepstral frame (indicating the presence of harmonic peaks as opposed to broadband noise). Combined with the ZCR, spectral rolloff, centroid and flux and their variances over a one-second window, they report an accuracy of 94.5% in discriminating speech from music in radio broadcasts, virtually independent of the classifier used.

Few approaches employ unsupervised dimensionality reduction, interpretable as a form of feature learning: Mesgarani et al. (2006) apply multilinear SVD to a high-dimensional biologically-inspired audio representation. On a corpus of pure speech, music, environmental and animal sounds, they report a speech detection accuracy of 100% using an RBF-kernel SVM. Izumitani et al. (2008) compress mel-spectral frames with PCA. They achieve an accuracy of 92% in discriminating pure speech from speech mixed with music using a GMM classifier.

To the best of my knowledge, none of the methods published in literature before mine perform more sophisticated feature learning, let alone using (mc)RBMs. On other tasks in the audio domain, however, these models have already proven useful: Dahl et al. (2010) and Mohamed et al. (2012) learn features for speech recognition from spectrograms and waveforms, respectively, and Lee et al. (2009b), Hamel and Eck (2010) and Schlüter and Osendorfer (2011) learn features for musical genre classification or music similarity estimation.

In conclusion, prior research focused on using existing audio features, or put much effort into designing new features by hand. While all authors report promising results, they all use different datasets, making it hard to valuate their methods – as an example, the music detector of Minami et al. (1998) only yielded about 56% accuracy on a larger corpus of Seyerlehner et al. (2007), compared to 90% on their own data. Besides, some approaches only discriminate pure speech from pure music (Saunders, 1996; Carey et al., 1999; Scheirer and Slaney, 1997; Mesgarani et al., 2006), and are likely to fail for the kind of mixed signals I am interested in. In this work, I investigate whether unsupervisedly learned features are competitive to hand-crafted features for music and speech detection.

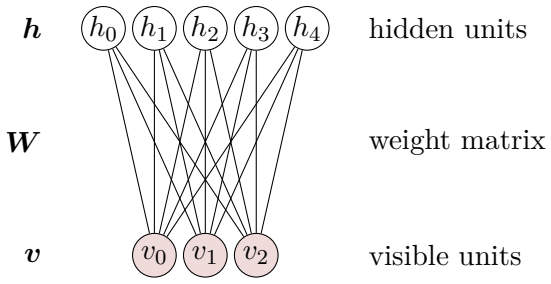


Figure 5.2: A Restricted Boltzmann Machine (RBM)

5.3 Feature Learning with mcRBMs

I will now introduce Restricted Boltzmann Machines (RBMs) and the variant used in this work, the mean-covariance Restricted Boltzmann Machine (mcRBM), as well as explain how they can be applied to audio data for both unsupervised feature learning and supervised classification. Note that since they do not play a central role for most of this thesis, I only give a compact description of the models here; for a more slow-paced introduction please see Krizhevsky (2009, pp. 6–16), Schlüter (2011, pp. 61–77), or Fischer and Igel (2012), for example.

5.3.1 Restricted Boltzmann Machines and Deep Belief Nets

An RBM (Smolensky, 1986) is an undirected graphical model consisting of visible units $\boldsymbol{v}$ representing observable data, and hidden units $\boldsymbol{h}$ giving a latent representation of the data. Visible and hidden units form two layers fully connected to each other, without within-layer connections – hence "restricted" (Figure 5.2).

The RBM defines a joint probability distribution of visible and hidden states via an energy function, such that configurations of low energy are more probable:

$$p(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\big(-E(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{\theta})\big), \tag{5.1}$$

where

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{u},\boldsymbol{g}} \exp\big(-E(\boldsymbol{u}, \boldsymbol{g}, \boldsymbol{\theta})\big) \tag{5.2}$$

is the normalizing *partition function*.[1] The energy function defines the space of possible energy surfaces – hence, the type of RBM –, and the model parameters $\boldsymbol{\theta}$, which include the connection weights, shape the energy surface.

[1] For reasonably sized models, the partition function is computationally intractable, as it requires enumerating all possible configurations. However, we will not need to compute it.

The most basic type of RBM restricts all unit states to be binary and uses the following energy function:

$$E_b(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{\theta}) = -\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h} - \boldsymbol{v}^T \boldsymbol{a} - \boldsymbol{h}^T \boldsymbol{b}, \tag{5.3}$$

where $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b})$ are the connection weights, visible and hidden bias terms, respectively. Inserting (5.3) into (5.1), we can derive the probability of a unit taking its "on" state (the unit's *activation*) conditioned on the other units:[2]

$$P(h_k = 1|\boldsymbol{v}, \boldsymbol{\theta}) = \sigma\Big(b_k + \sum_i v_i W_{ik}\Big) \tag{5.4}$$

$$P(v_k = 1|\boldsymbol{h}, \boldsymbol{\theta}) = \sigma\Big(a_k + \sum_j W_{kj} h_j\Big), \tag{5.5}$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function. As the activation of a hidden unit h_k only depends on the visible units, all hidden activations can be computed in parallel. Thus, determining the latent representation $\boldsymbol{h}$ of a data point $\boldsymbol{v}$ in this model is equivalent to passing it through a feedforward network with logistic units, then sampling binary states from the activations. Likewise, constructing a data point from a latent representation amounts to passing it back through the same network, again with a logistic activation function and binary sampling. The weight matrix $\boldsymbol{W}$ thus plays a double role: Its columns either act as feature detectors or as templates for generating data, each controlling or controlled by a single hidden unit.

Training an RBM means adjusting $\boldsymbol{\theta}$ such that $p(\boldsymbol{v}|\boldsymbol{\theta})$, the marginal probability density of the visible units, approximates the observed distribution of a set of training data; this is equivalent to maximizing the likelihood of the model under the data. Gradient ascent on the log likelihood yields a simple learning rule for a single connection weight (bias rules are similar):

$$W_{ij} \leftarrow W_{ij} + \eta\Big(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}\Big), \tag{5.6}$$

where η denotes the learning rate, $\langle v_i h_j \rangle_{\text{data}}$ is the (unnormalized) correlation of a visible and hidden unit in the training data, and $\langle v_i h_j \rangle_{\text{model}}$ is the same correlation under the model's data distribution. Considering Equation 5.3, each update lowers the energy for training data and raises the energy in low-energy regions. The first correlation can be easily computed by applying Equation 5.4 to each training data point. For the second correlation, we need to sample data points from the model. Directly sampling $p(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{\theta})$ is intractable due to the partition function, but we can apply *Gibbs sampling*: Starting from a random configuration, sampling $\boldsymbol{h}$ and $\boldsymbol{v}$ in

[2] For the derivation, see Krizhevsky (2009, p. 9) or Schlüter (2011, p. 143), for example.

turns (Equations 5.4 and 5.5) runs a Markov chain which, when converged, produces samples from the model. To make learning more efficient, Hinton (2002) proposed to start the chain at actual training examples and run it for a small number of k steps only (*Contrastive Divergence* learning, or CD-k).

After training, an RBM will produce samples resembling the training data when starting from a randomized state and performing Gibbs sampling for long enough. For this to work, the weights $\boldsymbol{W}$ must have learned useful templates to generate (parts of) data points – i.e., typical features found in the data. Thus, an RBM's hidden representation of a data point is an abstract description in terms of typical features, which makes it attractive for unsupervised feature extraction.

To obtain even more abstract representations, we can train an RBM on the latent representations of another RBM, learning features of features that capture higher-order correlations in the data. Recursively applying this principle, this creates a stack of RBMs termed a Deep Belief Net (DBN) (Hinton and Salakhutdinov, 2006).

5.3.2 The mean-covariance Restricted Boltzmann Machine

As the type of RBM discussed above has binary visible units, its generative model $p(\boldsymbol{v}|\boldsymbol{\theta})$ cannot approximate non-binary data, and thus will not learn useful features for non-binary inputs.

With a slight change of the energy function, though, we obtain an RBM capable of modelling real-valued data, which I will refer to as *mRBM*:

$$E_m(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{\theta}) = -\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h} + \frac{1}{2}(\boldsymbol{v} - \boldsymbol{a})^T(\boldsymbol{v} - \boldsymbol{a}) - \boldsymbol{b}^T \boldsymbol{h} \tag{5.7}$$

Its latent representations still follow Equation 5.4, but the distribution of visible states conditioned on the hidden states becomes:[3]

$$p(\boldsymbol{v}|\boldsymbol{h_m}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{v}|\boldsymbol{a} + \boldsymbol{W}\boldsymbol{h_m}, \boldsymbol{I}), \tag{5.8}$$

where $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the multivariate Gaussian probability density function with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Constructing a data point from a latent representation is thus equivalent to passing it through a one-layer network with *linear* output units, then adding isotropic Gaussian noise of unit standard deviation.

However, independent Gaussian noise does not yield a good generative model for most real-world data. To take into account pairwise dependencies of input variables, a third-order RBM can be defined, with weights $W_{i,j,k}$ connecting hidden units h_k to *pairs* of visible units v_i, v_j. By factorizing and tying these weights (Memisevic and Hinton, 2010; Ranzato et al., 2010), parameters can be reduced to a filter matrix

[3] For the derivation, see Krizhevsky (2009, p. 13) or Schlüter (2014, p. 6), for example.

$\boldsymbol{C}$ connecting the input twice to a set of *factors* and a pooling matrix $\boldsymbol{P}$ mapping factors to hidden units (Figure 5.3b). The energy function is

$$E_c(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{\theta}) = -(\boldsymbol{v}^T\boldsymbol{C})^2\boldsymbol{P}\boldsymbol{h} - \boldsymbol{c}^T\boldsymbol{h}, \tag{5.9}$$

yielding

$$p(h_k = 1|\boldsymbol{v}, \boldsymbol{\theta}) = \sigma\Big(\boldsymbol{c} + \big((\boldsymbol{v}^T\boldsymbol{C})^2\boldsymbol{P}\big)^T\Big) \tag{5.10}$$

$$p(\boldsymbol{v}|\boldsymbol{h_c}, \boldsymbol{\theta}) = \mathcal{N}(0, (\boldsymbol{C}\mathrm{diag}(\boldsymbol{P}\boldsymbol{h_c})\boldsymbol{C}^T)^{-1}). \tag{5.11}$$

Computing the latent representation now corresponds to passing the data through a *two-layer* feedforward net with a squared activation function in the hidden layer, and usual sigmoid output units. Constructing a data point can no longer be interpreted as a feedforward net: Instead, the data point is sampled from a Gaussian with a covariance matrix depending on the hidden unit states. However, this Gaussian is restricted to have zero mean.

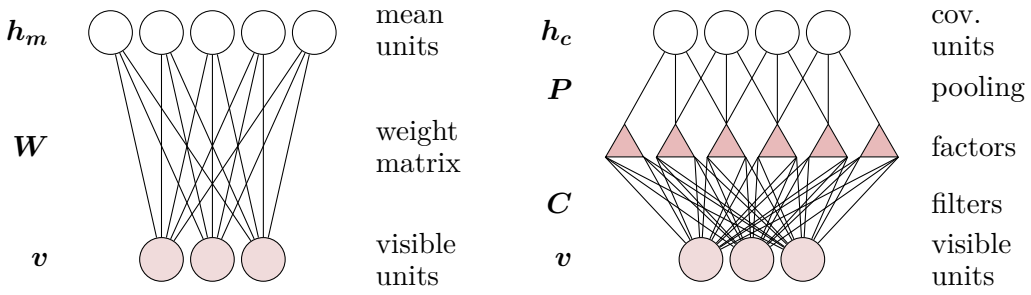
The mean-covariance Restricted Boltzmann Machine (Ranzato and Hinton, 2010) combines the former two models by adding their energy functions:

$$E_{mc}(\boldsymbol{v}, \boldsymbol{h_m}, \boldsymbol{h_c}, \boldsymbol{\theta}) = E_m(\boldsymbol{v}, \boldsymbol{h_m}, \boldsymbol{\theta_m}) + E_c(\boldsymbol{v}, \boldsymbol{h_c}, \boldsymbol{\theta_c}) \tag{5.12}$$

$p(\boldsymbol{v}|\boldsymbol{h_m}, \boldsymbol{h_c}, \boldsymbol{\theta})$ becomes the product of the two original Gaussian distributions, resulting in a generative model of two types of hidden units (Figure 5.3). It explains each datapoint as a linear combination of templates in $\boldsymbol{W}$, selectively smoothed with filters in $\boldsymbol{C}$. mcRBMs can still be trained with Contrastive Divergence, using a different sampling method to avoid the matrix inversion of Equation 5.11. To make learning more robust, input data and filters in $\boldsymbol{C}$ are normalized to unit L_2 norm when computing the hidden covariance unit states, and the pooling matrix $\boldsymbol{P}$ is constrained to a topographic mapping (associating a small group of neighbouring factors to each hidden covariance unit, as in Figure 5.3b), kept nonpositive and normalized to unit L_1 norm. For more details on the model and its training procedure, we refer the reader to Ranzato and Hinton (2010); Ranzato et al. (2010).

5.3.3 Discriminative Fine-Tuning

Using RBMs for feature extraction means computing their latent representation for given input data, usually omitting the binary sampling step. As explained in Sections 5.3.1 and 5.3.2, this can always be interpreted as passing the input data through a feedforward network, or Multi-Layer Perceptron (MLP), with either sigmoid or squared transfer functions. Thus, instead of merely training a classifier on the RBM's representations, we can add another layer on top and train the full network for classification with backpropagation, gently tuning the existing feature



(a) mRBM (b) cRBM

Figure 5.3: Diagrams of the two parts of a mean-covariance RBM.

detectors to the task at hands. Unsupervised learning can then be seen as a pre-training step finding a good initialization for backpropagation (Erhan et al., 2010) – especially for the lower layers, which are most strongly affected by the diminishing gradient effect (i.e., the effect that the error signal of the output layer gets weaker and weaker when backpropagated through the network, see p. 34).

5.3.4 Application to Audio Data

Originally, mcRBMs have been designed to model patches of natural images. Here, I follow Dahl et al. (2010) and Schlüter and Osendorfer (2011) and apply it to excerpts of mel spectrograms, allowing it to model local time-frequency structure in sounds.

Specifically, the input data is converted to 22.05 kHz monaural signals, followed by a Short-Time Fourier Transform (STFT) with a 1024-sample Hann window and a hop size of 512 samples. The magnitude spectrum of each frame is passed through a bank of 70 triangular filters equally spaced on the mel scale, covering the range from 50 Hz to 6854 Hz, and a log function is applied (resulting in log-magnitude mel spectral frames). Consecutive frames are joined to form overlapping blocks of 39 frames each, with a step size of 1 frame (i.e., 38 frames overlap). Each block covers about 0.93 s of audio, assumed sufficient to detect music or speech. Since the model is oblivious to the spatial layout of the spectral blocks, we can treat them as vectors for further preprocessing. Again following Dahl et al. (2010) and Schlüter and Osendorfer (2011), blocks are decorrelated with PCA, compressed to 99% of their original variance by omitting the least significant principal components (in this case, from 2730 to 1606 dimensions), and whitened by dividing each component by its standard deviation – these steps free the model from having to learn the overall covariance of spectrogram bins. The resulting vectors form the input data for the mcRBM, mapping each component to a visible unit.

5.4 Experimental Results

I performed a range of experiments to evaluate my approach. In this section, I will describe the dataset used in my experiments, the different variants of my method as well as approaches by other authors I evaluated, and then perform a qualitative analysis of the features learned as well as a quantitative analysis in terms of classification performance.

5.4.1 Dataset

The dataset consists of 42 hours of radio broadcasts finely segmented (with a resolution of 200 ms) into speech/nonspeech and music/nonmusic sections by paid students. 30 hours have been recorded from web streams of 6 Swiss radio stations in segments of 30 minutes randomly distributed over the course of a week, to capture as many different shows as possible. The chosen radio stations (DRS Virus, RSI Rete 2, RSR Couleur 3, Radio Central, Radio Chablais, RTR Rumantsch) range from indie rock to classical music and cover the four official languages of Switzerland: Swiss German, French, Italian and Rumantsch. The remaining 12 hours have been captured from lower-bitrate web streams of 4 Austrian radio stations (Ö1, Ö3, FM 4, Life Radio) as continuous 3-hour recordings, again covering different music styles and two languages: Austrian German and English.

15 hours of the Swiss recordings were used for training, another 6 hours for validation (and tuning hyperparameters) and the remaining 9 hours for testing. The Austrian recordings served as an additional test set to evaluate robustness to different recording conditions and generalization to unseen radio stations.

5.4.2 Evaluated Methods

I will now detail the architecture and training procedure of the network, describe reduced variants for control experiments and introduce three approaches by other authors evaluated on the same corpus.

5.4.2.1 This work

The full system consists of an mcRBM of 256 mean units and 1296 factors mapped to 324 covariance units (the smaller of the two architectures in Schlüter and Osendorfer, 2011), with two binary RBMs of 512 and 256 hidden units stacked on top. The mcRBM was trained unsupervisedly on spectral blocks extracted from the training set as detailed in Section 5.3.4. I trained it for 50 epochs on 453,120 training examples split into mini-batches of 128 data points with a learning rate of 0.02, L_1 weight decay of 0.001 and pooling matrix P constrained to a 2D topographic mapping. Subsequently, the RBM of 512 hidden units was trained on the mcRBM's

latent representations for 100 epochs with a learning rate of 0.01, L_1 weight decay of 0.0001 and momentum 0.9 (reduced to 0.45 for the first 20 epochs, following Hinton, 2010), linearly switching from CD-1 to CD-15 during training to counter the decreasing mixing rate in Gibbs sampling. The second RBM was trained on the first RBM's representations using the same settings.

Speech and music detection were treated as two separate classification problems handled by two separately fine-tuned instances of the network. For fine-tuning, I added a single sigmoid output unit and trained the resulting network on the full training set of 2,321,280 spectral blocks each paired with the binary label at its centre. Training was performed by backpropagation with cross-entropy error, a learning rate of 0.01, and momentum 0.9 (reduced to 0.45 for the first 10 epochs, then raised in steps during the next 10 epochs). Each network was trained for 100 epochs, monitoring the classification error at threshold 0.5 on the validation set. The epoch of lowest validation classification error was selected for evaluation on the test sets.

As the networks' block-wise predictions tend to be noisy, I apply a sliding median filter[4] to the sequence of network outputs on a file before thresholding the values to obtain binary decisions. By optimization on the validation set, I set the filter length to 250 frames (5.8 s) for music and 100 frames (2.3 s) for speech.

To understand by how much each component of the system influences the final result, I created four reduced variants in addition to the full system:

1. *MLP on mel:* To see how a classifier performs directly on the low-level audio representation, I trained a Multi-Layer Perceptron (MLP) of 512 and 256 hidden units on the whitened mel-spectral blocks.

2. *P on mcRBM:* In a second step, I trained a single Perceptron on the mcRBM output, to assess how useful the unsupervisedly learned features are to a linear classifier.

3. *MLP on mcRBM:* I repeated the same with an MLP of 512 and 256 hidden units.

4. *DBN on mcRBM:* I stacked the 512-unit and 256-unit RBMs on top of the mcRBM and fine-tuned them, still leaving the mcRBM unchanged as an unsupervised feature extractor (as in Dahl et al., 2010). This tests whether pretrained RBMs outperform randomly initialized weights (variant 3).

5. *DBN incl. mcRBM:* The final system includes the mcRBM in supervised fine-tuning with backpropagation.

[4] Compared to a sliding average, median filtering has the advantage of not blurring clearly localized decision boundaries; a step function remains unaffected by a running median filter.

All networks were trained on an Nvidia GTX 580 GPU with cudamat (Mnih, 2009), using the code provided by Ranzato and Hinton (2010) for the mcRBMs, and my own implementations for the binary RBMs and MLPs. Pretraining took 8 h for the mcRBM, 45 and 23 minutes for the two RBMs, respectively. 100 epochs of fine-tuning took 10 h when backpropagating through the full network, and 1.5 h when tuning the two RBMs only.

5.4.2.2 MFCCs

As a simple baseline, I extract 40 MFCCs, their first order derivative (delta) and second order derivative (acceleration) using yaafe (Mathieu et al., 2010) – these features have shown good results in Carey et al. (1999); Pinquier et al. (2002). I normalize features by subtracting the mean and dividing each dimension by its standard deviation (both determined on the training set), then train two MLPs of 512 units in the first hidden layer and 256 units in the second hidden layer for speech and music detection, respectively. I use the same training parameters as for the mcRBM-based system, and post-process the predictions with the same sliding median filters.

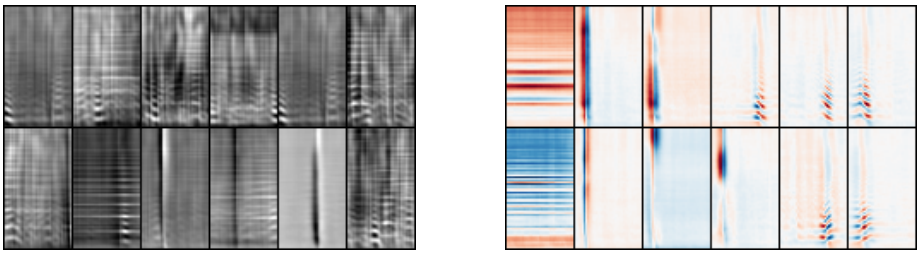
5.4.2.3 Liu et al.

Liu et al. (2007) extract a set of 8 standard audio features along with their variances over a short window, resulting in a 94-dimensional feature vector per audio frame. With a small MLP of 10 hidden units, they report near-perfect results of 98% F-Score for music and speech detection. Here, I extract the same set of features using yaafe (Mathieu et al., 2010). To rule out any influence of the classifier, I then process the feature vectors just like the MFCCs above.[5]

5.4.2.4 Seyerlehner et al.

Seyerlehner et al. (2007) engineered a feature for robust music detection in the presence of speech or noise, and demonstrated its performance on a corpus of TV recordings. It is based on the detection of horizontal structures in the spectrogram, i.e., sustained tones typical for music, and outputs a single value per timestep. I extract 5 such values per second, and apply a sliding median filter of 5.8 s as for the other approaches. By design, this feature is only useful for music detection, but it is especially interesting for the qualitative analysis of learned features in the next subsection.

[5] Note that my MLP is considerably larger than the 10-unit MLP of Liu et al. (2007). Control experiments with smaller networks show similar or worse performance. This is consistent with empirical results of Caruana et al. (2001) indicating that shallow large networks do not tend to overfit when trained with backpropagation and early stopping, as done here.



(a) mean units (unwhitened $\boldsymbol{W}$ columns) (b) factors (unwhitened $\boldsymbol{C}$ columns)

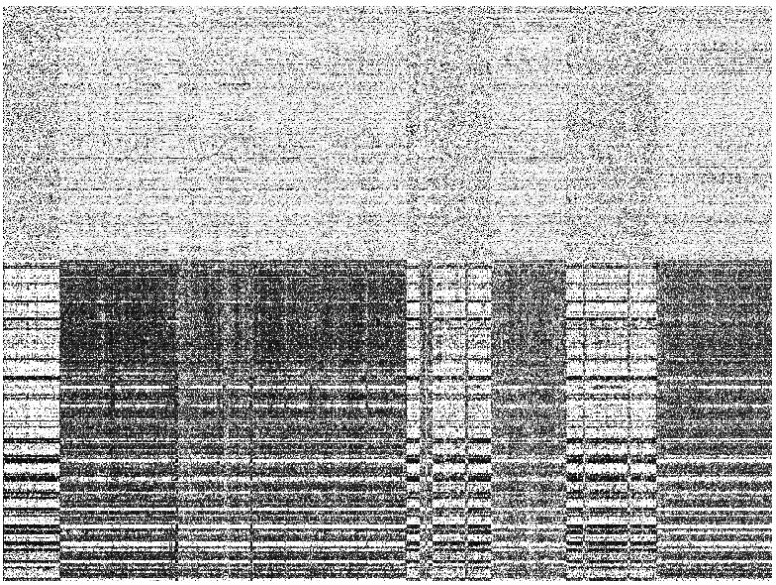
Figure 5.4: Exemplary features learned by the mcRBM. Each block represents 929 ms of a spectrogram: Time increases from left to right, mel-frequency from bottom to top. Mean units (black/white indicating small/large values) activate when the dot product of template and spectrogram patch is large. Factors (red/blue indicating negative/positive values) activate when the *squared* dot product is large, i.e., when there is a large difference between energy falling into the negative and positive bins of the template.

5.4.3 Learned Features

In Figure 5.4, we see a random selection of filters learned by the mcRBM *before discriminative fine-tuning*. Each block shows the unwhitened incoming weights of a mean unit (Figure 5.4a) or factor (Figure 5.4b), matching the layout of a one-second spectrogram excerpt. Most filters exhibit distinct horizontal or vertical patterns, some even display structures faintly resembling formants in speech.

To better understand the filters, I plotted the activations of all hidden units over the course of a 30-minute test set recording (Figure 5.5a). Remarkably, the mcRBM's latent representations show quite clearly which sections are dominated by music and which are dominated by speech (cf. Figures 5.5b, 5.5c). Looking closely at the activations of the hidden covariance units (lower part of Figure 5.5a), we can see units that are active during speech and inactive during music, and other units that behave the opposite.

In Figure 5.7, we zoom into a 10-seconds excerpt of the file. Figure 5.7a shows the mel-scaled spectrogram used as input for the mcRBM (the same as in Figure 5.1). It starts with a few seconds of pure speech and continues with music, clearly visible in the spectrogram. For the last few seconds, the radio host speaks again, with faint background music (barely visible). Figure 5.7b plots the corresponding activations of two covariance units that displayed roughly opposite behaviour on the whole file, and Figure 5.7c shows the filters of factors connected to these units. Mind that the factors connect to the covariance units through *negative* weights $\boldsymbol{P}$ (Section 5.3.2),

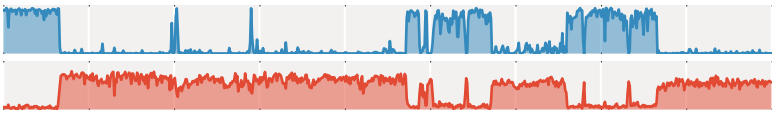


(a) Latent representation of the mcRBM. Each row depicts a hidden unit's activation over the course of the broadcast, mean units in the upper part, covariance units in the lower (generally darker) part.

(b) Music ground truth

(c) Speech ground truth

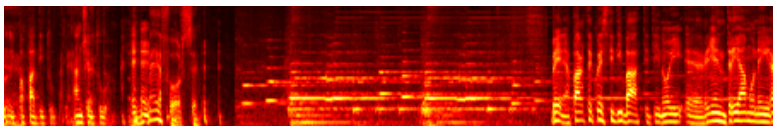
Figure 5.5: Unsupervisedly learned representation and ground truth for a 30-minute radio broadcast. Time proceeds from left to right.



(a) Activations over time of two selected hidden covariance units

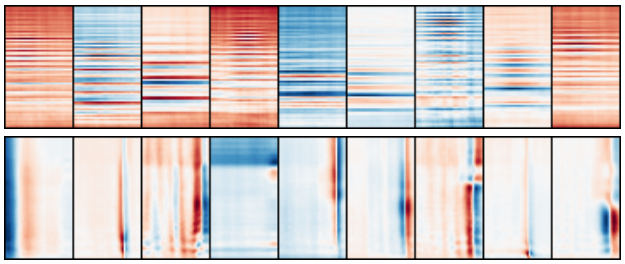
(b) Music detection feature of Seyerlehner et al. (2007)

Figure 5.6: Comparison of two hidden units' activations and an engineered music detector for the recording of Figure 5.5. The first unit acts approximately inversely to the music detector.



(a) Mel spectrogram, frequency increasing from bottom to top

(b) Activations over time of two selected hidden covariance units



(c) Factors negatively connected to the two covariance units

Figure 5.7: Spectrogram and activations of two covariance units for a 10-second excerpt of pure speech, pure music, and speech with faint background music. The top unit is inactive at sustained notes, the bottom unit is inactive at sudden loudness changes.

so an active factor results in an *inactive* covariance unit.[6] Considering this, we can see that the first unit is an anti-detector for sustained tones present in music (missing most background music, though), and the second unit's filters react on quick changes of loudness as occurring in speech (but also on strong musical onsets).

As we learned in Section 5.2, similar characteristics have also been addressed by engineered features for music (Hawley, 1993; Minami et al., 1998; Seyerlehner et al., 2007) and speech detection (Scheirer and Slaney, 1997). Exemplarily, we plot the feature of Seyerlehner et al. (2007) in Figure 5.6b. By our reasoning, as it detects horizontal structure, it should respond contrarily to the first covariance unit. Figure 5.6a shows that this is roughly the case, including dips in one curve where the other curve peaks, so the mcRBM seems to have re-invented their feature.

Note that these features were obtained with purely unsupervised learning – at this stage, the mcRBM has not seen any class labels, nor was it trained to find a binary segmentation. Merely the structure of the data drove it to develop two different sets of features for generating speech and music.

[6] See Ranzato et al. (2010, Sec. 5) for why implementing the model this way is advantageous.

5.4.4 Classification Results

To assess how useful the features are for the task of speech and music detection, we will now evaluate their classification performance on the two test sets. Specifically, I compute the frame-wise real-valued predictions of each system, smooth them as detailed in Section 5.4.2 and then apply a binary thresholding to obtain a label for each frame. Comparing the labels to ground truth provided by our annotators, I count the number of true positives tp, false positives fp, true negatives tn and false negatives fn. From these statistics, I compute four standard evaluation metrics: Accuracy $(tp+fn)/(tp+fp+tn+fn)$, precision $tp/(tp+fp)$, recall $tp/(tp+fn)$ and F-score: $2 \cdot \text{precision} \cdot \text{recall}/(\text{precision}+\text{recall})$.

For each method, I list results using a neutral threshold of 0.5, and a higher threshold of 0.7 which trades recall for higher precision. Predictions of Seyerlehner et al. are not limited between zero and one, so here I report results for the best thresholds (in terms of accuracy and F-score) on the validation set (0.85) and test sets (0.7) instead.[7]

Table 5.1 shows results for speech detection. Focusing on the Swiss test set, we can see that a linear classifier on features learned by an mcRBM (*P on mcRBM*) already performs quite well, but is inferior to what a discriminatively trained MLP learns from the same low-level spectrogram representation (*MLP on mel*). However, nonlinear classifiers on mcRBM features outperform both MFCCs and the feature set of Liu et al., and fine-tuning the mcRBM (*DBN incl. mcRBM*) brings an additional boost in performance. Interestingly, pre-training RBMs is not better than random initialization (*DBN on mcRBM* vs. *MLP on mcRBM*). On the Austrian test set, all classifiers perform worse, indicating that either they are slightly overfitted to Swiss radio stations, or they suffer from the reduced recording quality. This is especially true for the fully fine-tuned network: On the Austrian broadcasts, it is inferior to untuned mcRBMs (*DBN incl. mcRBM* vs. *MLP/DBN on mcRBM*).

For music detection, results look a little different (Table 5.2). On the Swiss test set, the *DBN* and *MLP on mcRBM* are outperformed by hand-crafted features, but the fully fine-tuned network *DBN incl. mcRBM* performs best by a large margin (2.7% misclassifications compared to 3.4% for the second best model). Results fit the observation that untuned mcRBM features miss background music. Curiously, the simple *MLP on mel* beats more complex feature sets – possibly because it sees a longer context. Again, all classifiers perform worse on the Austrian test set, including Seyerlehner et al. which was never trained on Swiss broadcasts, suggesting that the Austrian set is generally more difficult. The mcRBM-based methods now outperform all others, with the fully fine-tuned *DBN incl. mcRBM* still performing best – overfitting to Swiss stations seems to be less of a problem for music.

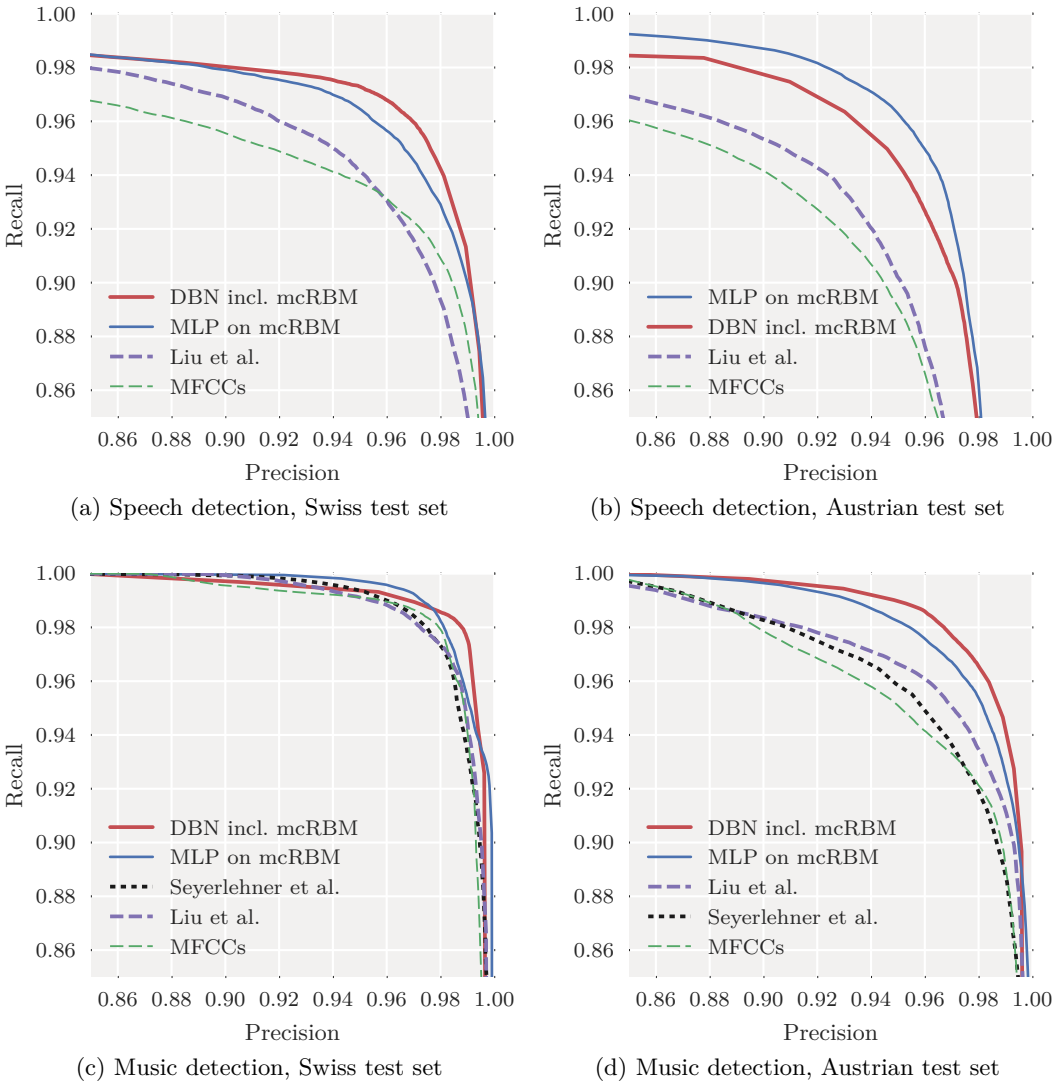
[7] Optimizing the threshold on the test sets is unfair, but shows that, on our corpus, the feature of Seyerlehner et al. is inferior to multi-feature approaches even with this radical measure.

		Swiss test set				Austrian test set			
	threshold	acc.	prc.	rec.	F-sc.	acc.	prc.	rec.	F-sc.
P on mcRBM	0.5	96.3	89.0	95.8	92.3	95.8	93.8	93.7	93.8
	0.7	97.0	94.0	93.0	93.5	95.1	96.0	89.4	92.6
MLP on mel	0.5	97.3	92.8	95.7	94.2	94.5	95.5	88.0	91.6
	0.7	97.6	95.0	94.4	94.7	93.8	96.3	85.1	90.4
MFCCs	0.5	97.3	94.2	94.1	94.1	94.6	95.3	88.5	91.8
	0.7	97.6	97.6	91.6	94.5	93.4	96.9	83.2	89.5
Liu et al.	0.5	97.4	93.5	95.4	94.4	95.2	94.7	90.9	92.7
	0.7	97.5	95.5	93.7	94.6	94.6	95.9	87.8	91.7
DBN on mcRBM	0.5	97.4	92.5	96.7	94.6	96.6	95.4	94.4	94.9
	0.7	97.8	95.4	95.1	95.3	96.0	96.7	91.3	93.9
MLP on mcRBM	0.5	97.9	93.9	97.1	95.4	**97.0**	95.9	95.1	**95.5**
	0.7	98.1	95.9	95.7	95.8	96.6	96.8	93.1	94.9
DBN incl. mcRBM	0.5	98.3	96.0	96.8	**96.4**	95.9	96.7	91.2	93.9
	0.7	**98.4**	96.4	96.5	**96.4**	95.8	97.0	90.3	93.6

Table 5.1: Speech detection performance of all methods on both test sets. For each method, we report the accuracy, precision, recall and F-score in percent at binarization thresholds of 0.5 and 0.7. The best accuracy and F-score per column are marked in bold.

		Swiss test set				Austrian test set			
	threshold	acc.	prc.	rec.	F-sc.	acc.	prc.	rec.	F-sc.
P on mcRBM	0.5	94.4	98.2	95.2	96.7	94.1	98.6	94.2	96.4
	0.7	94.1	99.1	93.9	96.4	91.6	99.2	90.5	94.7
MFCCs	0.5	95.6	98.5	96.2	97.4	92.1	96.8	93.4	95.1
	0.7	94.4	99.0	94.3	96.6	92.0	98.0	92.1	95.0
DBN on mcRBM	0.5	95.7	98.6	96.2	97.4	94.7	97.6	95.8	96.7
	0.7	94.8	99.0	94.8	96.8	93.9	98.5	93.9	96.2
MLP on mcRBM	0.5	95.9	98.7	96.4	97.5	94.7	97.8	95.7	96.7
	0.7	95.1	99.1	95.1	97.0	93.9	98.6	93.9	96.2
Seyerlehner et al.	0.7	96.1	97.9	97.4	97.7	92.4	94.9	95.9	95.4
	0.85	93.8	99.0	93.6	96.2	89.0	99.2	87.2	92.9
Liu et al.	0.5	96.1	98.1	97.3	97.7	93.5	96.8	95.2	96.0
	0.7	95.6	98.7	96.1	97.4	93.1	97.9	93.6	95.7
MLP on mel	0.5	96.6	98.7	97.3	98.0	94.2	95.9	97.0	96.5
	0.7	96.0	98.9	96.3	97.6	93.9	96.9	95.7	96.3
DBN incl. mcRBM	0.5	**97.3**	98.7	98.1	**98.4**	**95.6**	97.0	97.7	**97.3**
	0.7	**97.3**	98.8	98.0	**98.4**	**95.6**	97.3	97.4	**97.3**

Table 5.2: Music detection performance of all method on both test sets.



(a) Speech detection, Swiss test set

(b) Speech detection, Austrian test set

(c) Music detection, Swiss test set

(d) Music detection, Austrian test set

Figure 5.8: Precision/recall curves for speech and music detection on the two test sets. Note the axis range; plots start at 85% precision and recall to be able to discern the methods.

In Figure 5.8, I additionally plot the precision/recall curves for Liu et al., Seyerlehner et al. (if applicable), our *MLP on mcRBM* and *DBN incl. mcRBM* on the two test sets. The curves show that our conclusions from the table are valid for a range of reasonable thresholds.

5.5 Extensions and Dead Ends

While the mcRBM learned suitable features for music and speech detection, it has a potential problem: Despite being a generative model, the mcRBM is not actually good at generating plausible spectrogram excerpts. Generative models were initially proposed for unsupervised pretraining (see Section 2.2.4.4, p. 34) precisely because of their generative capabilities (Hinton, 2007), and it seemed plausible that a better generative model would also learn better features for discriminative tasks. One way to improve this would be designing the model to facilitate generating spectrograms – such as by including convolutions –, another way would be to make it more expressive, either by clever design – such as the mcRBM compared to a standard RBM –, or by simply increasing its depth.

The first route was taken by Lee et al. (2009a), who trained three-layer convolutional DBNs on images to learn a hierarchy of object parts.[8] While natural images are different from spectrograms, they share properties making them suitable for convolutional processing (see Section 4.2.2, p. 66). However, shortly before the work presented in this chapter, Dieleman et al. (2011) trained a two-layer convolutional DBN on chroma and MFCC-like features for artist, genre and key classification, with only marginal improvements through unsupervised pretraining. I decided to focus on increasing the depth, deferring the convolutional aspect for now.

My goal was to learn a deeper feature hierarchy for audio data with a better generative model, either for the task of this chapter or for other music analysis tasks requiring a higher level of abstraction (such as an extension of my earlier work on music similarity estimation using mcRBMs in Schlüter and Osendorfer, 2011). The easiest way – training RBMs on top of the mcRBM to form a DBN – did not seem fruitful, given that a randomly-initialized MLP on top of the mcRBM features outperformed the DBN in Section 5.4.4. So I implemented and experimented with a range of extensions, some of which had just been developed around that time:

Deep Boltzmann Machines (DBMs): Instead of only training a stack of RBMs layer by layer, a DBM (Salakhutdinov and Hinton, 2009a) is a deep generative model that can train all layers jointly. In practice, it is much more difficult to train than an RBM, which can be circumvented by pre-training it layerwise as a DBN (Salakhutdinov and Hinton, 2009a, 2012).

Improved sampling: Different authors proposed various way to improve sampling in an RBM or DBM, which in turn would improve training (through better estimates of the gradient in Equation 5.6): Persistent contrastive divergence (Tieleman, 2008), Fast weights (Tieleman and Hinton, 2009), Coupled adaptive simulated tempering (Salakhutdinov, 2010).

[8] Lee et al. (2009a, Fig. 3) famously shows how the second layer learns eyes, mouths and noses from the first-layer edges and blobs, assembled to faces by the third layer.

Centring trick: Montavon and Müller (2012) proposed a reparameterization that allows DBMs to be trained from scratch, without pretraining. It is based on mean-centring hidden units, much similar to the later technique of batch normalization for feedforward networks (Ioffe and Szegedy, 2015).

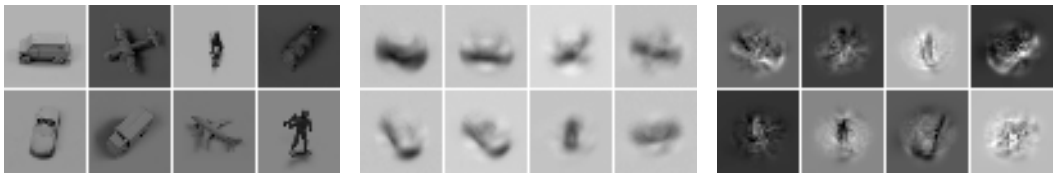
Improved formulations: The mRBM of Equation 5.7 is commonly referred to as a Gaussian-Bernoulli RBM, since it has Gaussian-distributed visible units and Bernoulli-distributed (binary) hidden units. In this formulation, the variance of visible units is fixed to 1, and it is trained on whitened data. However, it can still be advantageous to have the model learn the variance used for sampling (without going as far as the mcRBM, which models a covariance depending on the hidden states). There are different formulations for including the variance (Krizhevsky, 2009, Eq. 1.8; Cho et al., 2011, Eq. 2; Goodfellow et al., 2013, p. 8), with different implications on the training dynamics and on including tempering or the centring trick.[9]

Regularization: Both RBMs and DBMs are usually *overcomplete*: Their hidden representation of an input has more dimensions than the input itself, so they could just learn to copy it verbatim. The only reason they still learn abstract features is the way they are trained. With selected modifications to the training procedure, it is possible to further influence the resulting features. A common approach is to regularize activations to produce sparse hidden activations. As a particularly promising approach, Goh et al. (2010) proposed to modify the hidden states in Equation 5.6 to induce sparsity both per example (using few features to explain a single data point) and across examples (encouraging features to specialize towards subsets of data points).

Deep Energy Models (DEMs): As an alternative to DBMs, Ngiam et al. (2011) proposed to replace the energy function governing the model (Equation 5.1) by a deep neural network. Trained using the same sampling technique as for mcRBMs, this allows much more freedom in designing the model.

I initially experimented with images rather than spectrograms – real-valued and spatially correlated, they pose similar challenges, but generated examples can be inspected and compared more easily and in large quantities. Specifically, I used the NORB dataset of greyscale toy objects (LeCun et al., 2004). With the extended arsenal of methods described above, I eventually managed to train models creating somewhat plausible images (Figure 5.9), but could not reproduce the fidelity of Salakhutdinov and Hinton (2009a, Fig. 5) and Ngiam et al. (2011, Fig. 5) (a problem also encountered by other authors such as Goodfellow et al., 2013, Sec. 5.2). Furthermore, they were still far from the size of typical spectrogram excerpts.

[9] I later published the detailed derivations in a technical report (Schlüter, 2014).



(a) input examples (b) DBM samples (c) DEM samples

Figure 5.9: Attempts at training generative models on 32×32 pixel photographs. Samples by a Deep Boltzmann Machine (DBM) of 1024 Gaussian visible units and two hidden layers of 400 and 100 binary units capture the basic shapes, but not the varying lighting conditions and are very blurry. Samples by a Deep Energy Model (DEM) of two PoT (Product of Student-t) layers of 600 units each, trained on PCA-whitened images compressed to 176 components, capture the lighting conditions, but have strong artefacts (the equivalence of blurring in principal space). Neither seemed suitable to be applied to spectrograms.

Eventually, when Krizhevsky et al. (2012) showed in December 2012 that unsupervised generative pretraining was not required at all to obtain deep neural networks with state-of-the-art discriminative performance, I abandoned this line of research.

5.6 Discussion

In this chapter, I showed how to train a mcRBM on spectrogram excerpts to learn audio features. Similar to how another feature learning approach unsupervisedly found that images of digits come in 10 different shapes (Ranzato et al., 2008), the mcRBM discovered that radio stations produce two different kinds of content that are to be modelled separately (the same lateralization of speech and music processing has also been observed in the human brain, see Tervaniemi and Hugdahl, 2003). Exploiting this, I was able to build a highly accurate speech and music detector outperforming hand-crafted features on a large corpus of recorded radio broadcasts.

With accuracies between 95% and 98%, the classifier is not perfect – for example, it misses background music if it is too faint, or mistakes rap parts in hip hop music for speech. Furthermore, it performed better on recordings from the same stations it was trained on than on lower-quality recordings of a different country. Since the classifier is based on machine learning, it should be possible to partly remedy this by using a larger and more diverse training set (including more difficult examples, and varying recording conditions). Distinguishing rap from natural speech would

require a much more sophisticated approach and larger context to recognize rhymes and flow, but could be circumvented by non-machine-learning approaches such as fingerprinting to identify known music pieces.

For large-scale applications, it could be interesting to evaluate if smaller models yield similar results at lower computational costs. Although the classifier works at about 50× real-time on a cheap consumer graphics card from 2012 and 45× real-time on a single core of a modern CPU (and has been put to use in its current form, see Appendix A), the architecture used in my experiments might be oversized for this classification problem.

If tasked with the same problem today, I would choose a different approach – the dataset is large enough to be solved with purely supervised learning, now that techniques for training deep networks without pretraining exist (Section 2.2.4.4, p. 33). In fact, out of curiosity, I later trained the CNN of Chapter 9 on the dataset of radio broadcasts and achieved about the same results – interestingly, results were not better with modern techniques, but could be reached more easily. For problems in which labelled examples are scarce, but unlabelled examples are easy to obtain, unsupervised learning is still a useful technique. However, in such cases, one probably would neither use RBMs nor a strictly separated pretraining/finetuning strategy, but more modern semi-supervised approaches such as pseudo-labelling (Lee, 2013) or Ladder Networks (Rasmus et al., 2015). And even for purely generative models, RBMs have by now been superseded by Variational Auto-Encoders (Kingma and Welling, 2014), autoregressive models (e.g., van den Oord et al., 2016b) or various variants of Generative Adversarial Networks (Goodfellow et al., 2014).

From today's perspective, the work presented in this chapter mostly serves as a neat demonstration of what unsupervised learning is capable of: Learning an almost perfect binary distinction between music and speech just from the data (Figure 5.5a).

6 Commercial-Scale Music Similarity Estimation

We will now turn our attention to the task of music similarity estimation: Given two recordings of music pieces, determine how similar they sound. In particular, we focus on using this for music recommendations: Given a large collection of music pieces and a query piece, determine the pieces that sound most similar. This is particularly interesting for commercial catalogues, as it provides a way to find songs in the unpopular "long tail" not reachable by collaborative filtering.

The problem addressed in this chapter is that state-of-the-art music similarity

measures are too slow for large databases, as they are based on comparing very high-dimensional or non-vector song representations that are difficult to index. By training deep neural networks to map such song representations to binary codes, it becomes possible to quickly find a small set of likely neighbours for a query to be refined with the original expensive similarity measure. I show that for commercial-scale databases and two state-of-the-art similarity measures, this outperforms six previous attempts at approximate nearest neighbour search. For the better of the two similarity measures, when required to return (on average) 90% of true nearest neighbours, the method developed in this chapter is expected to answer 4.2 queries per second for the nearest neighbour or 1.3 queries per second for the 50 nearest neighbours to a song among a collection of 30 million songs using a single CPU core; an up to 260 fold speedup over a full scan of 90% of the database.

This work has previously been published in Schlüter (2013). This chapter mostly follows this publication, with a modified introduction, an additional method and results I previously omitted for space restrictions, more figures and two new final sections. The remainder of this chapter is organized as follows: Section 6.1 specifies the task, Section 6.2 links it to existing work on approximate nearest neighbour search, and Section 6.3 establishes what to focus on when designing a method for approximate nearest neighbour search. Section 6.4 introduces the two music similarity measures chosen for the experiments, and Section 6.5 introduces the eight methods used to accelerate them (including two newly proposed for this task). Section 6.6 describes the experiments and results, Section 6.7 presents unpublished follow-up work, and Section 6.8 concludes with a discussion from today's perspective.

6.1 Introduction

Content-based music similarity measures allow to scan a collection for songs that *sound* similar to a query, and could provide new ways to discover music in the steadily growing catalogues of online distributors. However, an exhaustive scan over a large database is too slow with state-of-the-art similarity measures, and many indexing methods are not applicable due to the high dimensionality of the search space. Fortunately, for music discovery, we do not necessarily need the true nearest neighbours to a query. By allowing results to deviate from the true neighbours, we enable the use of fast methods for approximate k nearest neighbour (k-NN) search.

Specifically, I consider the following scenario: (1) We have a commercial-scale music collection, (2) when answering queries, we want to return on average at least a fraction Q of the items an exhaustive scan would find, and (3) we cannot afford costly computations when a song enters or leaves the collection (ruling out nonparametric methods, or precomputing all answers). I then search for the fastest retrieval method under these constraints.

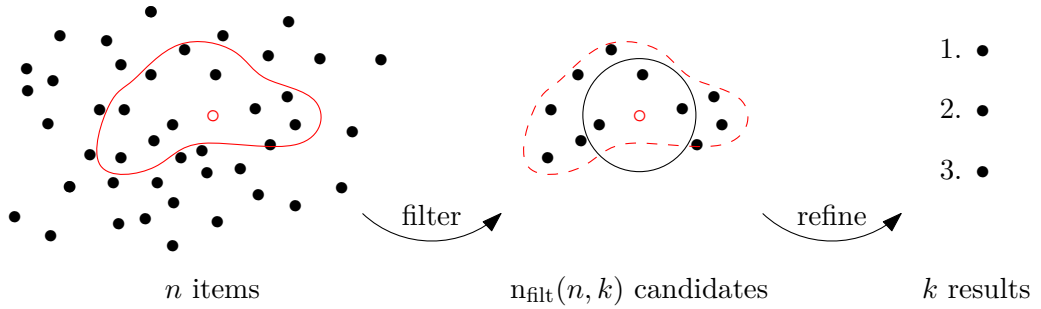


Figure 6.1: The filter-and-refine approach to accelerate nearest neighbour search: Instead of comparing a query (red hollow dot) with all n other items (filled dots) to find the k nearest neighbours, a prefilter finds a smaller set of candidates (filter step) the query is compared to (refine step).

Some approximate k-NN search methods cannot return exactly k items for a query, and some would miss the recall constraint. To circumvent this, I will use a *filter-and-refine* approach (Schnitzer et al., 2009): The approximate k-NN search is used as a fast prefilter returning a small subset of the collection, which is then refined to the k best items therein using the original similarity measure. Figure 6.1 illustrates this concept.

Compared to existing work on approximate k-NN search, what makes this quest special is the nature of state-of-the-art music similarity measures, which can be high-dimensional, non-vectorial or non-metric, and a low upper bound on database sizes: The largest online music store only offers 26 million songs as of February 2013,[1] while web-scale image or document retrieval needs to handle billions of items. We will see that this allows us to even use methods of linear time complexity.

6.2 Related Work

Among the first approaches to fast k-NN search were space partitioning trees (Bentley, 1975). However, their basic forms degrade to an exhaustive search for high-dimensional vector spaces (e.g., Weber et al., 1998, Sec. 3.2). McFee and Lanckriet (2011) use PCA spill trees (an extension of k-d trees) on 222-dimensional models of 890,000 songs, reporting a 120 fold speedup over a full scan when missing 80% of true neighbours. No comparison to other methods is given.

Hash-based methods promise cheap lookup costs. Cai et al. (2007) apply Locality-Sensitive Hashing (LSH) (Gionis et al., 1999) to 114,000 songs, but do not evaluate k-NN recall. Torralba et al. (2008) learn binary codes with Restricted Boltzmann

[1] `http://www.apple.com/pr/library/2013/02/06iTunes-Store-Sets-New-Record-with-25-Billion-Songs-Sold.html`, accessed May 2017

Machines (RBMs) for 12.9 mio. images, achieving 80% 50-NN recall looking at 0.2% of the database and decisively outperforming LSH. Using similar techniques, other researchers learn codes for documents (Salakhutdinov and Hinton, 2009b) and images (Krizhevsky and Hinton, 2011; Norouzi et al., 2012a), but, to the best of my knowledge, never for songs.

Pivot-based methods map items to a vector space only using distances to landmark items, allowing to index non-vectorial data. Rafailidis et al. (2011) apply L-Isomap to 9,000 songs, not evaluating k-NN recall. Schnitzer et al. (2009) apply FastMap to 100,000 songs, achieving 80% 10-NN recall looking at 1% of the collection. Chávez et al. (2005) propose a method for mapping items to byte vectors, obtaining 90% 1-NN recall examining 10% of the data.

In what follows, I will adapt the most promising methods for application on two music similarity measures and evaluate their performance under our scenario.

6.3 Filter-Refine Cost Model

The cost of answering a query using a filter-and-refine approach can be decomposed into the cost for running the prefilter and the cost for refining the selection to k items:

$$\text{cost}_{\text{filtref}}(n, k) = \text{cost}_{\text{filt}}(n, k) + \text{cost}_{\text{ref}}(\text{n}_{\text{filt}}(n, k), k), \tag{6.1}$$

where $\text{n}_{\text{filt}}(n, k)$ is the number of candidates returned by the prefilter (Figure 6.1), which can be tuned to reach a particular true nearest neighbour recall.

We assume that the refine step is a linear scan over these candidates, picking the k best:

$$\text{cost}_{\text{ref}}(n_f, k) = n_f \cdot (R + \log(k) \cdot S), \tag{6.2}$$

where R is the cost for computing the distance of a query to a candidate, here determined by the choice of a music similarity measure, and $\log(k) \cdot S$ is the cost for updating a k-element min-heap of the best candidates found so far.[2]

The cost of a full linear scan over the database is equal to the refine cost with a defunct prefilter:

$$\text{cost}_{\text{full}}(n, k) = \text{cost}_{\text{ref}}(n, k) \tag{6.3}$$

However, in our scenario, the filter-and-refine method is only required to return on average a fraction Q of the true neighbours, so a better baseline is to apply the similarity measure to a random fraction Q of the dataset. This can be expressed as a zero-cost prefilter returning a fraction Q of the dataset:

$$\text{cost}_{\text{baseline}}(n, k) = \text{cost}_{\text{ref}}(Q \cdot n, k) \tag{6.4}$$

[2] This model is not entirely correct, as the heap is generally not updated for each item. However, for $k \leq 100 \ll n_f$, I empirically found the sorting cost to be indeed linear in n_f, which is all we need for our argument.

Under these assumptions, using a prefilter gives the following speedup factor over the baseline:

$$\begin{aligned}
\mathrm{spu}(Q,n,k) &= \frac{\mathrm{cost}_{\mathrm{baseline}}(n,k)}{\mathrm{cost}_{\mathrm{filtref}}(n,k)} = \frac{\mathrm{cost}_{\mathrm{ref}}(Q\cdot n,k)}{\mathrm{cost}_{\mathrm{filtref}}(n,k)} \\
&= \frac{Q\cdot \mathrm{cost}_{\mathrm{ref}}(n,k)}{\mathrm{cost}_{\mathrm{filt}}(n,k)+\mathrm{cost}_{\mathrm{ref}}(\mathrm{n}_{\mathrm{filt}}(n,k),k)} \\
&= Q\cdot\left(\frac{\mathrm{cost}_{\mathrm{filt}}(n,k)+\mathrm{cost}_{\mathrm{ref}}(\mathrm{n}_{\mathrm{filt}}(n,k),k)}{\mathrm{cost}_{\mathrm{ref}}(n,k)}\right)^{-1} \\
&= Q\cdot\left(\frac{\mathrm{cost}_{\mathrm{filt}}(n,k)}{\mathrm{cost}_{\mathrm{ref}}(n,k)}+\frac{\mathrm{n}_{\mathrm{filt}}(n,k)}{n}\right)^{-1} \\
&= Q\cdot\left(\rho_{\mathrm{t}}(n,k)+\rho_{\mathrm{s}}(n,k)\right)^{-1} \qquad (6.5)
\end{aligned}$$

We see that making the prefilter fast compared to a full scan (small ρ_{t}) is just as important as making the filter selective (small ρ_{s}). More specifically, we need to minimize the sum of the two ratios to maximize the speedup factor. It follows that whenever the ratios differ by more than a factor of two, we gain more by decreasing the larger ratio to match the smaller one than we can possibly gain by decreasing the smaller ratio.

As we will see in the experiments, some existing methods put too much emphasis on a fast prefilter, resulting in $\rho_{\mathrm{t}}(n,k)$ being orders of magnitude smaller than $\rho_{\mathrm{s}}(n,k)$ especially for large databases. In this work I will balance the two ratios better to maximize the speedup factor.

6.4 Music Similarity Measures

For the experiments, I chose two very different similarity measures: One based on high-dimensional vectors, and another based on Gaussian distributions.

6.4.1 Vector-Based Measure

Seyerlehner et al. (2010b) propose a set of six *Block-Level Features* (see p. 61) to represent different aspects of a song's audio content, totalling in 9448 dimensions. These features work well for genre classification and tag prediction (Seyerlehner et al., 2010c), and similarity measures based on them ranked among the top three algorithms in the MIREX Audio Music Similarity (AMS) tasks 2010–2012.[3] For the similarity measure, the six feature vectors are individually compared by Manhattan

[3] http://www.music-ir.org/mirex/wiki/2010:Audio_Music_Similarity_and_Retrieval_Results, accessed May 2017

distance, and the resulting feature-wise distances are combined to form the final similarity estimation.

To combine the feature-wise distances, they must be brought to a similar scale. Instead of finding six appropriate scaling factors on an arbitrary dataset and fixing them, Seyerlehner et al. (2010b, Sec. 4) normalize the feature-wise distance matrices specifically for the dataset at hands. This *Distance Space Normalization (DSN)* processes each distance matrix entry by subtracting the mean and dividing by the standard deviation of its row and column (counting $D_{n,m}$ twice):

$$\mu_{n,m} = \frac{1}{2N}\left(\sum_{i=0}^{N-1} D_{i,m} + \sum_{j=0}^{N-1} D_{n,j}\right) = \frac{1}{2N}\sum_{i=0}^{N-1} D_{m,i} + D_{n,i} \tag{6.6}$$

$$\sigma_{n,m} = \frac{1}{2N-1}\sum_{i=0}^{N-1} (D_{m,i} - \mu_{n,m})^2 + (D_{n,i} - \mu_{n,m})^2 \tag{6.7}$$

$$\widehat{D}_{n,m} = (D_{n,m} - \mu_{n,m}) / \sigma_{n,m} \tag{6.8}$$

The six normalized matrices are added up and normalized once again to form the final similarities. When it is infeasible to compute full distance matrices, the statistics can be approximated from the distances to a fixed random subset of the collection and stored with each feature vector (Schnitzer, 2011, Sec. 4.3.2.2).

While the normalizations seem unnecessarily complex for the mere purpose of combining distances, Flexer et al. (2012) showed that they remove *hubs* – items appearing as neighbours of undesirably many other items, especially in high-dimensional space – and are vital to achieve state-of-the-art results. However, they also invalidate many assumptions on the distances: While still symmetrical, they do not correspond to vectorial distances any longer, and are not even metrical.

6.4.2 Gaussian-Based Measure

As a second method, I use the timbre model proposed by Mandel and Ellis (2005): Each song is represented by the mean vector and covariance matrix of its frame-wise Mel-Frequency Cepstral Coefficients (MFCCs, p. 58). Specifically, I use frames of 46 ms with 50% overlap, 37 Mel bands from 0 Hz to 11025 Hz and retain the first 25 MFCCs. Song distances are computed as the symmetrized Kullback-Leibler divergence between these multivariate Gaussian distributions (Schnitzer, 2007, p. 24), and normalized with DSN.

This measure does not reach state-of-the-art performance on its own, but forms the main component of the similarity measure by Pohle et al. (2009), which ranked among the top two algorithms in the MIREX AMS tasks 2009–2012. Furthermore, it is easy to reproduce and allows direct comparison to indexing experiments by Schnitzer et al. (2009) and Schnitzer (2011).

6.5 Indexing Methods

I will evaluate eight different methods for fast k-NN search: Five based on compressing or indexing the song models and three based on song similarities. Each method can be tuned to trade speed for improved selectivity.

6.5.1 Locality-Sensitive Hashing (LSH)

For vectors in an Euclidean space, the family of projections onto a random line, followed by binary thresholding or fixed-width quantization, is *locality-sensitive*: For such projections, two close items are more probable to be mapped to the same value than two items far apart (Gionis et al., 1999).

LSH uses $L \cdot K$ projections to map each item x_i to L discrete K-dimensional vectors $h_l(x_i)$. Using L conventional hash-table lookups, it can quickly find all items x_j matching a query q in at least one vector, $\exists_{l \leq L} h_l(q) = h_l(x_j)$.

Here, this serves as a prefilter for finding neighbour candidates. Increasing K makes it more likely for candidates to be true nearest neighbours, but strongly reduces the candidate set size. Increasing L makes the candidate set larger, but increases query and storage costs. As a complementary way to increase the number of candidates, Multi-probe LSH (Lv et al., 2007) considers items with a *close* match in one of their vectors.

6.5.2 Principal Component Analysis (PCA)

PCA finds a linear transformation $\boldsymbol{y} = \boldsymbol{W}^T \boldsymbol{x}$ of Euclidean vectors $\boldsymbol{x_i} \in \mathcal{D}$ to a lower-dimensional space minimizing the squared reconstruction error $\sum_i \|\boldsymbol{x_i} - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{x_i}\|_2^2$.

Nearest neighbours in the low-dimensional space are good candidates for neighbours in the original space, so a linear scan over items in the low-dimensional space serves as a natural prefilter. The candidate set size can be tuned at will to achieve a target k-NN recall. Increasing the dimensionality of the space allows to reduce the candidate set size, but increases prefilter costs.

6.5.3 Iterative Quantization (ITQ)

ITQ (Gong and Lazebnik, 2011) finds a rotation of the PCA transformation minimizing squared reconstruction error after bit quantization of the low-dimensional space: $\sum_i \|\boldsymbol{x_i} - \boldsymbol{W}\, \mathrm{b}(\boldsymbol{W}^T\boldsymbol{x_i})\|_2^2$, where $\mathrm{b}_i(\boldsymbol{z})$ is 1 for positive z_i and 0 otherwise.

It can serve as a prefilter just like PCA, but using bit vectors reduces computational costs for the linear scan. For compact bit codes, neighbours within a small Hamming distance of a query can alternatively be found with a constant number of conventional hash table lookups enumerating the Hamming ball around the query.

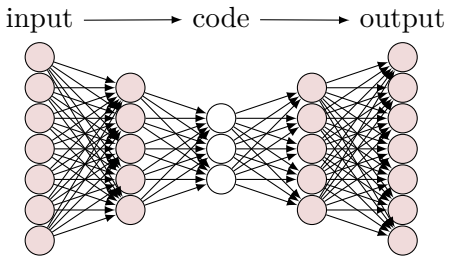


Figure 6.2: An Auto-Encoder is a network trained to compress its input down to few dimensions or bits and reconstruct it. It often has a symmetric architecture with successively smaller and then successively larger layers.

6.5.4 PCA Spill Trees

K-d Trees (Bentley, 1975) are binary trees recursively partitioning a vector space: Each node splits the space at a hyperplane, assigning the resulting half-spaces to its two child nodes. Similar items often reside in the same half-space, enabling efficient nearest neighbour search, unless they happen to be separated by one of the hyperplanes. Spill Trees (Liu et al., 2005) allow the half-spaces to overlap, making it less likely for close items to be separated. McFee and Lanckriet (2011) additionally propose to choose hyperplanes perpendicular to a dataset's principal components, and to strongly restrict the depth of the tree. In the resulting *PCA Spill Tree*, each item ends up in one or more leaves, with similar items often sharing at least one leaf. Locating the leaves for an item is linear in the database size (McFee and Lanckriet, 2011, Sec. 3.6), but can be avoided by precomputing all leaf sets.

Like McFee and Lanckriet (2011), I regard all items in the leaf set of a query to be candidate neighbours. The candidate set size can be increased by decreasing the tree depth or by increasing the overlap at each node.

6.5.5 Auto-Encoder (AE)

An AE finds a nonlinear transformation of inputs to a low-dimensional or binary code space and back to the input space, minimizing the difference between inputs and reconstructions (e.g., L_2 distance for Euclidean input vectors). Similar to PCA and ITQ, candidate neighbours to a query can quickly be found in the code space.

The transformation is realized as an artificial neural network – here, a Multi-Layer Perceptron (MLP) – and can be optimized with backpropagation. Figure 6.2 illustrates the architecture. For deep networks, it is helpful to initialize the network weights using Restricted Boltzmann Machines (RBMs). Salakhutdinov and Hinton (2009b) were the first to use a deep AE for approximate nearest neighbour search, under the term *Semantic Hashing*, and describe the method in detail.

6.5.6 Hamming Distance Metric Learning (HDML)

HDML (Norouzi et al., 2012a) finds a nonlinear transformation to a binary code space optimized to preserve neighbourhood relations of the input space. Specifically, for any triplet (x, x^+, x^-) of items for which x is closer to x^+ than to x^- in the input space, it aims to have x closer to x^+ than to x^- in the code space. Again, the transformation is realized as an artificial neural network, optimized with backpropagation, and HDML can be used as a prefilter just like ITQ or AE.

6.5.7 FastMap

FastMap (Faloutsos and Lin, 1995) maps items to a d-dimensional Euclidean space based on their (metric) distances to d previously chosen pivot pairs in the input space. Schnitzer et al. (2009) show how to apply FastMap to Gaussian-based models and propose an improved pivot selection strategy I will adopt.

FastMap serves as a prefilter like PCA, but supports non-vector models as it is purely distance-based. However, it assumes metric distances.[4] While the Manhattan distance of the vector-based measure is metrical, the symmetrized Kullback-Leibler divergence (sKLD) of the Gaussian-based measure is not, as it violates the triangle inequality. Schnitzer (2011, Sec. 5.3.4.1) recommend to scale the sKLD with $\log(1+x)$ to have it respect the triangle inequality almost always and improve FastMap results (better than the $\sqrt{x}$ scaling recommended by Schnitzer et al., 2009, Sec. 3.3.1), which I adopt. Since the DSN (Equation 6.8) would invalidate metric properties, I apply FastMap to the unnormalized distances. For the vector-based measure, this requires an alternative way to combine the six feature-wise distances. As done by Schnitzer (2011, Eq. 5.4/5, p. 105) in an equivalent situation, I normalize each feature by the average of standard deviations over distance matrix rows.

6.5.8 Permutation Index

A Permutation Index (Chávez et al., 2005) maps items to d-dimensional discrete vectors giving the ranks of d previously chosen pivots in terms of distance to the items. That is, each vector is a permutation of the set $\{0, 1, \ldots, d-1\}$. Permutations are compared by *Spearman's Footrule*, which equals the cheaply computable L_1 distance of the discrete vectors. Since vectors are discrete, they can be stored more compactly and compared more quickly than real-valued vectors.

This serves as a prefilter like FastMap, but does not necessarily require metric or near-metric distances. Since Chávez et al. only experimented with metrics, I still apply it to the unnormalized distances. The $\log(1 + x)$ scaling of sKL divergences can be ignored since it does not change the distance ranks.

[4] It can produce reasonable results with non-metric distances (Athitsos et al., 2004), but works better with metrics (Schnitzer, 2011, Fig. 5.2a).

6.6 Experimental Results

I will now compare the eight indexing methods empirically, conducting retrieval experiments for both similarity measures.

6.6.1 Dataset and Methodology

From a collection of 2.5 million 30-second song excerpts used by Schnitzer et al. (2009); Schnitzer (2011), I randomly select 120k albums of 120k different artists. 10k albums (124,013 songs) are used for training, 20k albums (246,117 songs) for validation and the remaining 90k albums (1,101,737 songs) for testing. In addition, I use 20k albums (253,347 songs) of the latter as a smaller test set.

For each applicable combination of similarity measure and indexing method, I will train different parameterizations of the method on the training set and determine the speedup over the baseline (Equation 6.5, p. 97) for retrieving on average 90% of the 1 or 50 nearest neighbours on the validation set. I will then evaluate the best parameterizations on the small test set to ensure I did not overfit on the validation set, and use the large test set to assess the methods' scalability.

Carrying out these evaluations requires a nontrivial amount of engineering. For example, obtaining the ground truth k-NN for the vector-based measure on the small test set requires computing Manhattan distances on $9448 \cdot 253347 \cdot 4\,\mathrm{B} \approx 9\,\mathrm{GiB}$ of features resulting in six distance matrices of $253347 \cdot 253348/2 \cdot 4\,\mathrm{B} \approx 120\,\mathrm{GiB}$ each (exploiting the symmetry), which need to be normalized by the row and column statistics (Equation 6.8), added up and normalized again, to finally compute the 50 nearest neighbours for each of the 253347 items. I solved this by implementing a separate algorithm for each step that does a single strictly sequential pass over the distance matrix or matrices written to or read from hard disk storage.
Moreover, naively implementing each of the eight methods for actual retrieval and then running 246117 validation set queries against them to compare results to the ground truth, with a wide range of different parameterizations to hit the 90% recall goal as exactly as possible, would be computationally infeasible. Instead I devised measures to efficiently compute the k-NN recall for different k and different candidate set sizes at once, tailored to the different indexing methods:

PCA, FastMap, Permutation Index: For methods based on vector distances, I precompute all vectors and then iterate over the queries, comparing the distances of the ground truth nearest neighbours to the n^{th} distance to record how many ground truth k-NN would have been retrieved with a candidate set of size n. This can be done for multiple n (and k) in parallel, obtaining a k-NN recall curve (as in Figure 6.3, p. 107) at the cost of a single distance matrix computation in mapping space. With a spline interpolation through the curve, I then estimate the candidate set size required for 90% recall.

ITQ, AE, HDML: For methods based on bit vector distances, distance values are discrete (the Hamming distance between l-bit vectors is in $[0, l]$), so often there is no unique way to obtain a candidate set of size n. Instead, I control the candidate set size via the radius of the Hamming ball around the query considered in code space. The k-NN recall per radius can be computed just from the Hamming distances of the ground truth nearest neighbours to their query items. With a single distance matrix computation in code space I then obtain the average candidate set size per Hamming radius.

PCA Spill Tree: For a spill tree, the candidate set for a query consists of all items of all leaves the query resides in. For efficient evaluation, I precompute the set of items per leave, and the set of leaves each item resides in. For each query, I compute the cardinality of the join of its leaves, obtaining the candidate set size, and the number of ground truth nearest neighbours whose leaf sets overlap with the query's leaf set, obtaining the k-NN recall. I reduce the tree depth by 1 to grow the candidate sets and repeat, obtaining a k-NN curve I can interpolate to estimate the candidate set size for 90% recall. As the number of leaves is relatively small, I express leaf sets as bit strings, so all operations except for the candidate set size estimation become cheap bit operations.

LSH: For all items, I precompute L projections of dimensionality K. For each query, I record which ground truth neighbours match the query in which of the L projections, to compute the k-NN recall for many L at once. In addition, I record how many items match the query in which projections, to estimate the candidate set size for each L. As this is still too slow, I compute the projections on GPU using cudamat (Mnih, 2009) and only use 10,000 and 250 queries to estimate the k-NN recall and candidate set sizes, respectively.

6.6.2 Vector-based Measure

To be able to compute the speedup, I first determine the costs of the similarity measure.[5] Computing 1 million 9,448-dimensional Manhattan distances takes 2.361 s, finding the (indices of) the smallest 100 distances takes 1.17 ms, and both costs scale linearly with the collection size, as assumed in Equation 6.2. Costs for the approximate DSN are negligible (see Section 6.4.1). For prefilters based on a linear scan, computing 1 million 80-dimensional L_2 distances takes 22 ms, and computing 1 million 1024-bit Hamming distances takes 9.8 ms. The costs of finding the best candidates in a linear scan depend on the candidate set size; we will use separate measurements for each case.

[5] All timings are reported on an Intel Core i7-2600 3.4 GHz CPU with DDR3 RAM, use a single core, and leverage AVX/POPCNT instructions. Implementations are in C, carefully optimized to maximize throughput: `http://jan-schlueter.de/pubs/phd/benchmark_dist.zip`

PCA: I start by evaluating PCA as a prefilter, as it proved useful as a preprocessing step for most other filters as well. To mimic how the similarity measure is combined from six features, I first apply PCA to each feature separately, compressing to about 10% of its size, then rescale each feature to unit mean standard deviation (this brings the distances to comparable ranges, and forms good inputs for the AE later) and stack the compressed features to form an 815-dimensional vector. Finally, I apply another PCA to compress these vectors to a size suitable for prefiltering.
In Table 6.1, we see that this cuts down query costs: For retrieving 90% of the true nearest neighbours, prefiltering with a linear scan over 40-dimensional PCA vectors takes $\rho_t = 0.56\%$ the time of a full scan and only needs to examine $\rho_s = 0.26\%$ of the database afterwards, resulting in a 110 fold speedup over the baseline ($0.9/(0.0056+0.0026)$, Equation 6.5). For retrieving 50-NN, it needs a larger candidate set, increasing the prefilter costs (due to higher sorting costs to find the candidates), but still achieving a 47 fold speedup. Varying the vector dimensionality changes the tradeoff between ρ_t and ρ_s, but does not improve the speedup.

LSH: I apply different versions of LSH to the 815-dimensional intermediate PCA representation.[6] First, I follow Slaney et al. (2012) to compute optimal quantization width, dimensionality and table count for 90% 1-NN recall under the assumption that all projections are independent (it suggests 92.192, 25 and 430, respectively). To reach our target 1-NN recall, I need a 3-fold increase in table count and obtain $\rho_s = 16.52\%$, which is not competitive. Turning to binary LSH, I fix the dimensionality K to 8, 16 or 20 bit and increase L until reaching 90% recall (for 20 bits and 50-NN, this needs 8353 hash tables). Even assuming zero prefilter costs, speedup is far below PCA. As a third alternative, I use a simple version of multi-probe LSH: I fix L and K, but consider all buckets within a Hamming distance of r to the query in any of the tables. I increase r to reach the target k-NN recall, still achieving moderate speedups of up to 25x only.

ITQ directly builds on the PCA transform above, but maps items to bit vectors. As discussed on the previous page, instead of directly tuning the candidate set size, I consider all items in a Hamming ball of radius r around the query (in code space), and tune r. This avoids the sorting costs for finding the candidates, so ρ_t becomes independent of the candidate set size. ITQ has small ρ_t, but large ρ_s, resulting in low speedups.

[6] PCA is a useful stepping stone as the DSN (Section 6.4.1) invalidates any theoretical guarantees of LSH finding the nearest neighbours in the original space. Directly working on the 9448-dimensional vectors, rescaling the six components to comparable range, consistently gave worse results.

	Method	1-NN ρ_t(%)	1-NN ρ_s(%)	1-NN spu	50-NN ρ_t(%)	50-NN ρ_s(%)	50-NN spu
PCA	20 dim	0.38	0.59	93x	0.58	1.85	37x
	40 dim	0.56	0.26	110x	0.71	1.20	47x
	80 dim	1.01	0.18	76x	1.16	1.12	40x
LSH	8 bit	0.00	17.23	5x	0.00	23.82	4x
	16 bit	0.00	6.66	14x	0.00	11.00	8x
	20 bit	0.00	4.68	19x	0.00	8.42	11x
mp-LSH	128x16 bit	0.83	11.12	8x	0.83	34.18	3x
	64x32 bit	0.83	6.03	13x	0.83	12.23	7x
	32x64 bit	0.83	3.65	20x	0.83	7.59	11x
	16x128 bit	0.83	3.58	20x	0.83	7.11	11x
	1x256 bit	0.10	3.85	23x	0.10	7.98	11x
	8x256 bit	0.83	2.80	25x	0.83	6.22	13x
ITQ	48 bit	0.02	7.32	12x	0.02	12.04	7x
	64 bit	0.03	5.43	16x	0.03	9.94	9x
	128 bit	0.05	4.74	19x	0.05	8.27	11x
	Spill Tree	0.00	10.25	9x	0.00	21.27	4x
AE	64 bit	0.03	2.14	41x	0.03	4.40	20x
	128 bit	0.05	0.57	144x	0.05	2.47	36x
	256 bit	0.10	0.24	**265x**	0.10	1.28	65x
	512 bit	0.21	0.14	258x	0.21	0.93	79x
	1024 bit	0.42	0.09	177x	0.42	0.70	**81x**
FastMap	20 dim	0.68	2.46	29x	1.12	5.22	14x
	40 dim	0.77	1.56	39x	1.11	3.72	19x
	80 dim	1.20	1.36	35x	1.53	3.43	18x
	128 dim	1.73	1.20	31x	2.03	3.07	18x

Table 6.1: Results for the vector-based music similarity measure on the validation set of 246,117 songs: Ratio of prefilter time to full scan (ρ_t), ratio of candidate set to dataset size (ρ_s) and resulting speedup over baseline (spu) for retrieving 90% of 1 and 50 true nearest neighbours, evaluated using all possible 246,117 queries.

PCA Spill Tree: I build a tree with spill factor 0.1 (the best performing of McFee and Lanckriet, 2011) and adjust the depth to reach our target recall. Assuming zero prefilter costs, it achieves poor speedups as it needs very large candidate sets to retrieve 90% of the true nearest neighbours.

AE: I train a deep AE on the 815-dimensional intermediate PCA representation, pretrained with stacked RBMs as done by Krizhevsky and Hinton (2011). I use an encoder architecture of 1024-256(-128(-64)) layers for the shorter codes, 1024-512 and 2048-1024 for the two longer codes, and a symmetric decoder.[7] I encourage binary codes by adding noise in the forward pass like Salakhutdinov and Hinton (2009b); especially for 128 bits and more, this worked better than thresholding like Krizhevsky and Hinton (2011). For 256 bits and less, it also helped to encourage zero mean code unit activations like Norouzi et al. (2012a, Eq. 12).
I use the learned codes as in ITQ. This gives a prefilter which is both faster than PCA and more selective, achieving a 265 fold speedup for 1-NN and 81 fold speedup for 50-NN queries. Note how the accuracy of longer codes pays off for 50-NN, while shorter codes win for 1-NN, with a good balance for 512-bit codes.

HDML learns codes from triplets of items (x, x^+, x^-), see Section 6.5.6. I select x^+ among the k^+ nearest neighbours of x, and x^- outside the 500 nearest neighbours. During training, I gradually increase k^+ from 10 to 200, to slowly increase the difficulty of the learning problem. I arrived at this scheme through careful exploration of different ideas, for both the vector-based and Gaussian-based measure. Instead of training a randomly initialized network like Norouzi et al. (2012a), I fine-tune the existing AEs. Unfortunately, this does not improve results compared to the AEs used as starting points. For example, for 128-bit codes, the best I could achieve was a 114-fold speedup for 1-NN (not included in table).

FastMap is about twice as fast as LSH or ITQ, but falls behind AE and PCA. Like PCA, it performs best at 40 dimensions.

Permutation Index: I compute a permutation index from 256 pivots (the 128 pivot pairs chosen for FastMap), the larger of the two pivot counts tried by Chávez et al. (2005). The results fall behind any of the other methods: Even with a candidate set size of 50% of the dataset, less than 80% of true 1-NN are retrieved (not included in table). This is better than a random prefilter, but not enough for our purposes.

[7] Results are robust to the exact choice of architecture as long as there is at least one hidden layer before the code layer, and the first hidden layer is wide enough.

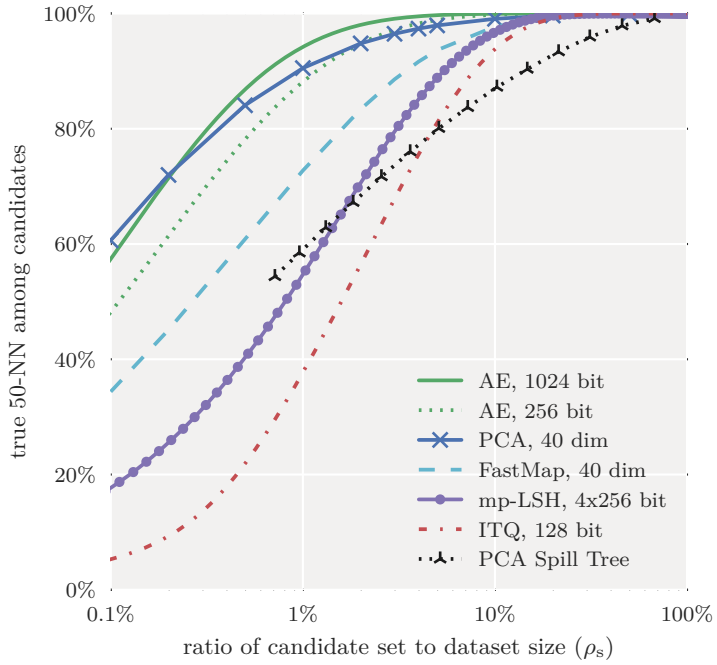


Figure 6.3: 50-NN recall versus candidate set size for the vector-based music similarity measure on the test set of 253,347 songs, averaged over all 253,347 possible queries.

Evaluating the best-performing instantiations of each method on the small test set, results are very similar to Table 6.1. As the relative prefilter costs ρ_t stay the same anyway, I only show how the candidate set size ρ_s and 50-NN recall interact (Figure 6.3). We can see that the 1024-bit AE again only needs about 0.7% of the dataset to find 90% of 50-NN, and we see that AE and PCA perform best over a wide range of target recall values. Besides, comparison with McFee and Lanckriet (2011, Fig. 4) shows that the PCA Spill Tree performs similar to its first publication. Extending the PCA Spill Tree curve further towards smaller candidate sets would require a deeper tree, but we can already see that it is not competitive.

6.6.3 Gaussian-based Measure

Again, I first determine the costs of the similarity measure: Computing 1 million symmetric Kullback-Leibler (sKL) divergences between 25-dimensional full-covariance Gaussian models takes 1.085 s, using precomputed inverse covariance matrices like Schnitzer (2007, Ch. 4.2). Note that most indexing methods evaluated above are vector-based and not applicable to Gaussian models, so I expect the most from HDML and FastMap, but still try AE and PCA to be sure.

AE: In order for learned codes to be useful, they must reflect the input space. For the input space at hands – data points are the mean vectors and covariance matrices of Gaussian distributions to be compared by sKL divergence –, it seems natural to learn codes by minimizing the sKL divergence between inputs $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ and reconstructions $\boldsymbol{\mu'}$, $\boldsymbol{\Sigma'}$, defined as (e.g., Li et al., 2005, Eq. 3):

$$sKL\Big(\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})\Big\|\mathcal{N}(\boldsymbol{\mu'},\boldsymbol{\Sigma'})\Big) = \frac{1}{4}\Big(\operatorname{trace}(\boldsymbol{\Sigma'}^{-1}\boldsymbol{\Sigma} + \boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma'}) + (\boldsymbol{\mu'}-\boldsymbol{\mu})^T(\boldsymbol{\Sigma}^{-1}+\boldsymbol{\Sigma'}^{-1})(\boldsymbol{\mu'}-\boldsymbol{\mu}) - 2N\Big) \quad (6.9)$$

As it is differentiable wrt. $\boldsymbol{\mu'}$, $\boldsymbol{\Sigma'}$, we can minimize it with a neural network. However, without further constraints, the network will learn to produce $\boldsymbol{\Sigma'}$ that are not positive definite and push the sKL unboundedly below zero. To avoid this, I express $\boldsymbol{\Sigma}$ by its Cholesky decomposition $\boldsymbol{C}$ and also interpret the network output as a Cholesky decomposition $\boldsymbol{C'}$, minimizing $sKL(\mathcal{N}(\boldsymbol{\mu},\boldsymbol{C}^T\boldsymbol{C})\|\mathcal{N}(\boldsymbol{\mu'},\boldsymbol{C'}^T\boldsymbol{C'}))$. To enforce $\boldsymbol{C'}^T\boldsymbol{C'}$ to be positive definite (not just semidefinite), I additionally apply the softplus function $\log(\exp(x)-1)$ to all diagonal entries $C'_{i,i}$.[8] Finally, this allows training an AE to reconstruct $\boldsymbol{\mu'}$, $\boldsymbol{C'}$ from $\boldsymbol{\mu}$, $\boldsymbol{C}$. Sadly, it only learns to reconstruct the centroid of all training data.

For a second attempt, I ignored all mathematical justification and trained an ordinary L_2-optimizing AE. Interestingly, this worked much better and also benefited from the modified input representation. Using the same architectures as in Section 6.6.2 and a similar preprocessing (I separately compress $\boldsymbol{\mu}$ and $\boldsymbol{C}$ with PCA to 99.9% variance, then scale to unit mean standard deviation), I obtain moderate speedups of up to 16x.

PCA on the same representation performs poorly.

HDML: I fine-tuned the existing AEs with HDML, as described for the vector-based measure. I obtain good results with 128-bit codes, but longer codes do not improve the speedup. To close the gap between ρ_t and ρ_s, I instead employ multiple 128-bit codes handled as in mp-LSH (i.e., a close match in at least one of the codes is a candidate), obtaining an up to 65-fold speedup.

FastMap is faster than AE, but slower than HDML. Results fall a bit behind Schnitzer et al. (2009) because I evaluate against nearest neighbours found with DSN, while they evaluate against neighbours on the original distances.

Permutation Index again works too poorly to be included in the table.

Again, Figure 6.4 demonstrates that the conclusions also hold for the test set and a wide range of target recall values.

[8] I later learned that almost the same idea for having a network output covariance matrices was proposed much earlier by Williams (1996), using $\exp(x)$ instead of softplus.

	Method	1-NN $\rho_t(\%)$	1-NN $\rho_s(\%)$	1-NN spu	50-NN $\rho_t(\%)$	50-NN $\rho_s(\%)$	50-NN spu
PCA	20 dim	6.18	16.03	4x	10.96	29.80	2x
	40 dim	6.00	14.04	4x	11.11	28.79	2x
AE	64 bit	0.06	11.89	8x	0.06	19.36	5x
	128 bit	0.11	6.95	13x	0.11	15.77	6x
	1024 bit	0.91	4.76	16x	0.91	13.13	6x
HDML	128 bit	0.11	1.46	57x	0.11	3.73	23x
	256 bit	0.23	1.37	56x	0.23	3.93	22x
	2x128 bit	0.23	1.15	**65x**	0.23	3.12	**27x**
	4x128 bit	0.45	1.09	58x	0.45	3.02	26x
	1024 bit	0.91	1.20	43x	0.91	4.65	16x
FastMap	20 dim	1.92	3.76	16x	3.50	8.31	8x
	40 dim	2.03	2.62	19x	3.37	6.48	9x
	80 dim	2.78	1.85	19x	3.85	4.94	10x
	128 dim	3.98	1.81	16x	5.03	4.85	9x

Table 6.2: Results for the Gaussian-based music similarity measure on the validation set of 246,117 songs.

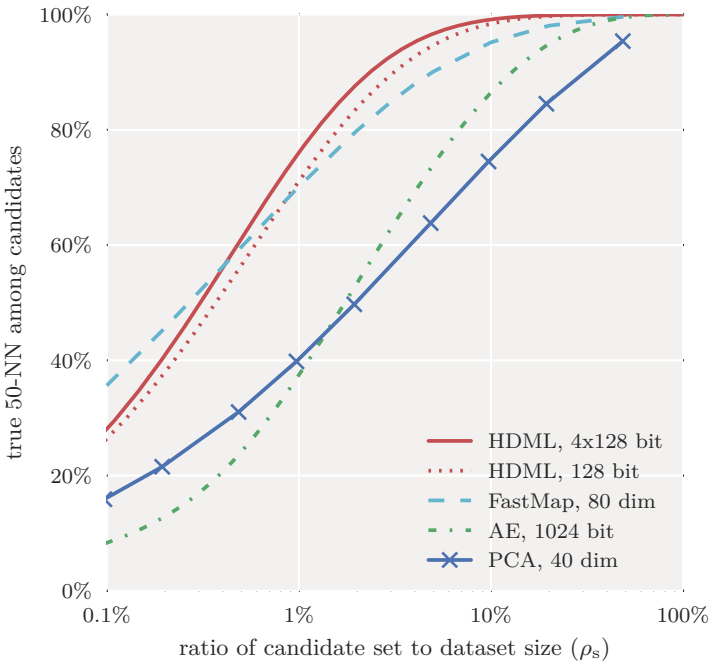


Figure 6.4: 50-NN recall versus candidate set size for the Gaussian-based music similarity measure on the test set of 253,347 songs.

6.6.4 Scalability

Finally, I evaluate how the best-performing approaches scale with the collection size. For the large test set of 1.1 million songs, it is computationally infeasible to compute the full distance matrix, which is needed to compute the exact DSN and the ground-truth k-NN (see Section 6.6.1). Thus, I have to change the means of evaluation.

For the DSN, I work around the distance matrix computation by using an estimation: Recalling Equation 6.8, to normalize the distance $D_{n,m}$ between two items n and m, we need the mean $\mu_{n,m}$ and standard deviation $\sigma_{n,m}$ over all distances $\boldsymbol{D}_{n,:}$ and $\boldsymbol{D}_{:,m} = \boldsymbol{D}_{m,:}$. First note that we can compute these statistics from two intermediate sums:

$$s_n = \sum_{i=0}^{N-1} D_{n,i} \qquad q_n = \sum_{i=0}^{N-1} D_{n,i}^2 \tag{6.10}$$

$$\mu_{n,m} = \frac{s_n + s_m}{2N} \qquad \sigma_{n,m}^2 = \frac{q_n + q_m - \frac{(s_n+s_m)^2}{2N}}{2N-1} \tag{6.11}$$

As this is too expensive, I instead estimate the intermediate sums from $Z < N$ distances:

$$\hat{s}_n = \frac{N}{Z} \sum_{i=0}^{Z-1} D_{n,i} \qquad \hat{q}_n = \frac{N}{Z} \sum_{i=0}^{Z-1} D_{n,i}^2 \tag{6.12}$$

Computing these estimates for all of the 1.1 million songs, we can normalize the distance for every pair of items. In practice, the estimates can be computed using any representative set of items, not necessarily the first Z items of the same collection – I use 10,000 items from the training set (one per album). Note that for the vector-based measure, we need to estimate separate sums for all six features, and additional sums for the second DSN stage applied to the combined normalized distances (which have to be normalized using the first-level estimates). I empirically verified that the approximation works reasonably well: On the validation set, for the vector-based measure, 908 of 1000 nearest neighbour queries produce the correct item, 79 return the second ground truth neighbour instead, and 13 the third. For the Gaussian-based measure, which only requires a single DSN estimation instead of seven, results are even better: 970 of 1000 nearest neighbour queries stay the same when estimating DSN parameters from 10,000 training items, 954 when estimating from 1000 training items, and 969 when estimating from 10,000 validation set items.

Since it is still infeasible to compute the ground truth k-NN for all 1.1 million items, I restrict the evaluation to 10,000 random queries. Again, I verified that this provides a reasonable estimate: On the validation set, the speedup factors determined for FastMap change by ± 1 when determined from 10,000 random queries instead of all 246,117 possible queries.

	Method	1-NN $\rho_t(\%)$	1-NN $\rho_s(\%)$	1-NN spu	50-NN $\rho_t(\%)$	50-NN $\rho_s(\%)$	50-NN spu
PCA	40 dim	0.55	0.16	126x	0.67	0.73	64x
PCA	80 dim	1.00	0.10	82x	1.12	0.64	51x
AE	256 bit	0.10	0.15	**351x**	0.10	0.74	107x
AE	512 bit	0.21	0.07	322x	0.21	0.46	**135x**
AE	1024 bit	0.42	0.05	193x	0.42	0.34	119x

Table 6.3: Results for the vector-based music similarity measure on the test set of 1.1 million songs, evaluated using 10,000 random queries.

	Method	1-NN $\rho_t(\%)$	1-NN $\rho_s(\%)$	1-NN spu	50-NN $\rho_t(\%)$	50-NN $\rho_s(\%)$	50-NN spu
HDML	128 bit	0.11	1.19	69x	0.11	2.62	33x
HDML	2x128 bit	0.23	0.93	**78x**	0.23	2.15	**38x**
HDML	4x128 bit	0.45	0.88	67x	0.45	2.08	36x
FastMap	20 dim	1.98	2.97	18x	3.42	6.09	9x
FastMap	40 dim	1.97	1.84	24x	3.17	4.44	12x
FastMap	80 dim	2.92	1.71	19x	4.09	4.24	11x
FastMap	128 dim	4.00	1.41	17x	5.08	3.76	10x

Table 6.4: Results for the Gaussian-based music similarity measure on the test set of 1.1 million songs, evaluated using 10,000 random queries.

The results for the two similarity measures are given in Tables 6.3 and 6.4. We see that all methods obtain a larger speedup compared to the validation set or the small test set: the candidate set sizes required to obtain 90% k-NN recall do not increase linearly with the dataset size, so ρ_s gets smaller, and the speedup over the baseline increases. We can infer that the methods scale better than linearly with the collection size.

To stay on the conservative side, I will still extrapolate linearly from the validation set to estimate the performance on a commercial-scale collection of 30 million songs: The best methods are expected to answer 4.2 1-NN queries or 1.3 50-NN queries per second with the vector-based measure, and 2.2 1-NN queries or 0.9 50-NN queries per second with the Gaussian-based one, using a single CPU core, with 90% true nearest neighbour recall.

Note that all these results assume that models are held in main memory. In practice, for large collections, this can be achieved by distributing queries over a cluster, but I will discuss some alternatives and more implications in Section 6.8.

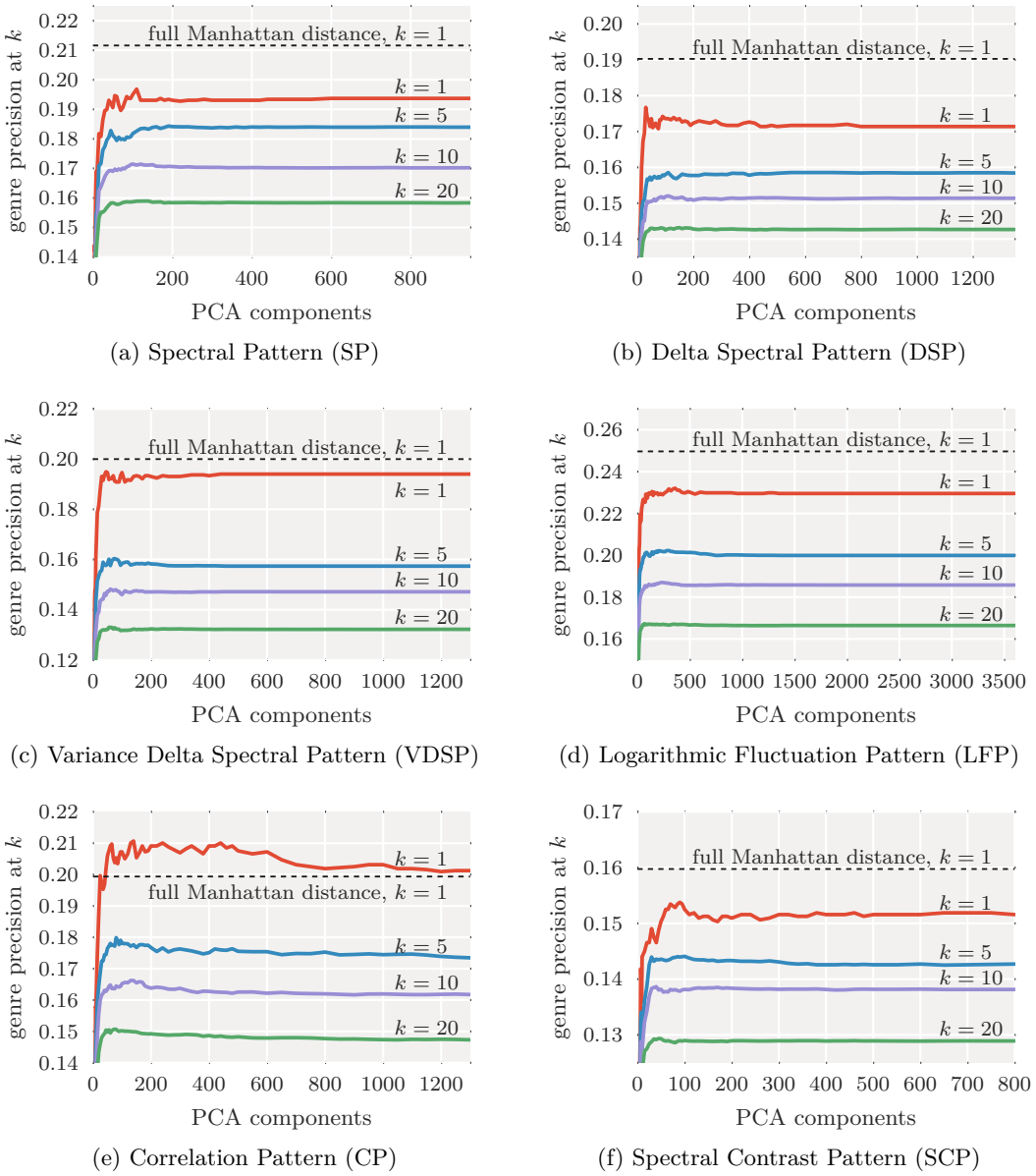
6.7 Extensions and Dead Ends

The vector-based measure performs much better than the Gaussian-based one, not only in terms of the speedup over the baseline, but also in absolute terms (see previous page). In practice, the gap will be even larger: The vector-based measure is a fully fledged state-of-the-art music similarity measure, while the Gaussian-based one only encompasses a timbre model, and would have to be combined with a rhythm model (which can also be Gaussian-based as in Pohle et al., 2009), slowing it down further. However, the vector-based measure comes with a catch: Storing a million song models takes $10^6 \cdot 9448 \cdot 4\,\mathrm{B} \approx 35.2\,\mathrm{GiB}$, and keeping them in fast enough memory to not stall the refine step can be challenging for large-scale collections. Thus, I investigated ways to compress the models without sacrificing performance. In particular, I tried three different approaches:

PCA: As explained in Section 6.5.2 (p. 99), with PCA, we can compress the feature vectors to a given target dimensionality in a way that minimizes the L_2 reconstruction error. The PCA transformation rotates the vector space to align each dimension with one of the data's principal components (the eigenvectors of its covariance matrix), and when using PCA for compression, we then omit the components along which the variance of the data is minimal. When comparing vectors with Euclidean distance, the PCA rotation does not change the distances, but dropping components does. To assess how many components we can omit for each of the six features, I compute the genre precision at k on the 1517-artist dataset by Seyerlehner et al. (2010b), using the Euclidean distance on a single PCA-compressed feature, for different k and numbers of components.[9] Figure 6.5 shows the results: Each feature can be compressed to about 1/10 of its size without affecting performance, except for the Correlation Pattern (Figure 6.5e), for which performance even improves. However, Seyerlehner proposed to compare features with Manhattan distance. For this measure, even the PCA rotation alone affects the distances between vectors. Figure 6.5 additionally shows the precision obtained with Manhattan distances on the uncompressed, unrotated original features. Except for the Correlation Pattern, they are noticeably higher than the corresponding precisions with Euclidean distances. We can of course rotate back the compressed features and compare them with Manhattan distance to regain lost precision,[10] but that is a high computational price to pay to shrink models.

[9] The genre precision at k is the fraction of k nearest neighbours to an item that share the item's genre label, averaged over all items in a collection. I use an artist filter, skipping neighbours of the same artist as the query (cf. Flexer, 2007). The same dataset and a similar evaluation has been used by Seyerlehner (2010, Sec. 5.3.3) to develop and optimize the six features.

[10] E.g., compressing the Logarithmic Fluctuation Pattern to 300 components and decompressing it gives a precision at $k = 5$ of 0.2150, compared to 0.2157 with the unmodified feature.



(a) Spectral Pattern (SP)

(b) Delta Spectral Pattern (DSP)

(c) Variance Delta Spectral Pattern (VDSP)

(d) Logarithmic Fluctuation Pattern (LFP)

(e) Correlation Pattern (CP)

(f) Spectral Contrast Pattern (SCP)

Figure 6.5: For each of the six features of the vector-based similarity measure, we see how the genre precision at k on the 1517-artists dataset is affected by compressing the feature with PCA, when comparing features with Euclidean distance. Features can be radically compressed without significantly deteriorating results. However, precision is higher when comparing the uncompressed features with Manhattan distance.

2D-DCT: Due to the way the six block-level features are computed (see p. 61), each has a two-dimensional topological structure, often with strong correlations between neighbouring entries. A possible way to compress them with explicit use of this structure – in contrast to PCA, which treats features as vectors – is to apply a two-dimensional Discrete Cosine Transform (2D-DCT). In a first attempt in collaboration with Dominik Schnitzer, we simply compressed each feature to one third of its size in each of the two dimensions (i.e., to one ninth of its total size), and then compared the compressed features with Manhattan distance. Later, I refined this by running a grid search to find a suitable number of components to keep for each feature, further restricting the search to have each feature end up with a dimensionality divisible by 8, in order to maximize throughput when computing feature-wise Manhattan distances with AVX instructions (which keep 8 floats of 32 bits per register, and load memory more efficiently when aligned on 32-byte boundaries). Specifically, I compressed the SP to 16×3, DSP to 16×11, VDSP to 48×4, LFP to 12×16, CP to 16×16, SCP to 18×4 components. Figure 6.6 shows the results in terms of genre precision at k for different datasets. Compared to the original similarity measure, even the simple compression scheme (labelled "DCT 1") works reasonably well, sometimes performing slightly worse and sometimes slightly better. The refined compression (labelled "DCT 2") slightly improves results over the basic one, and actually outperforms the uncompressed features on all but one dataset. This gives a much more practical similarity measure: For a million songs, it only requires 3.5 GiB of storage, and features can be directly compared, without decompressing them first.

AE: While a 2D-DCT uses the topological structure, it does not make any use of the data distribution at hands – while I tuned the number of components on training data, it might pay off to train a more complex model to compress the features. In fact, this is precisely what I did in this chapter, so an interesting follow-up question to the main experiments is: Can we directly use the binary codes learned by a deep neural network for retrieval, omitting the refine step? In Section 6.6, we always compared results for the binary codes against results by the full model in terms of overlapping nearest neighbours (a choice I will discuss in Section 6.8), not with an independent measure of quality such as the genre precision at k. In this scenario, we can use the slowest AE producing 1024-bit codes, since it is still many times faster than the full measure. I evaluate it on the same datasets as the two DCT-compressed variants, using Hamming distance on the codes followed by DSN. Figure 6.6 shows that it performs worse than, but close to the original measure. On the ballroom dataset, it is even better: Possibly, it does not strike the same balance between the six features, and is more strongly influenced by Fluctuation Patterns.

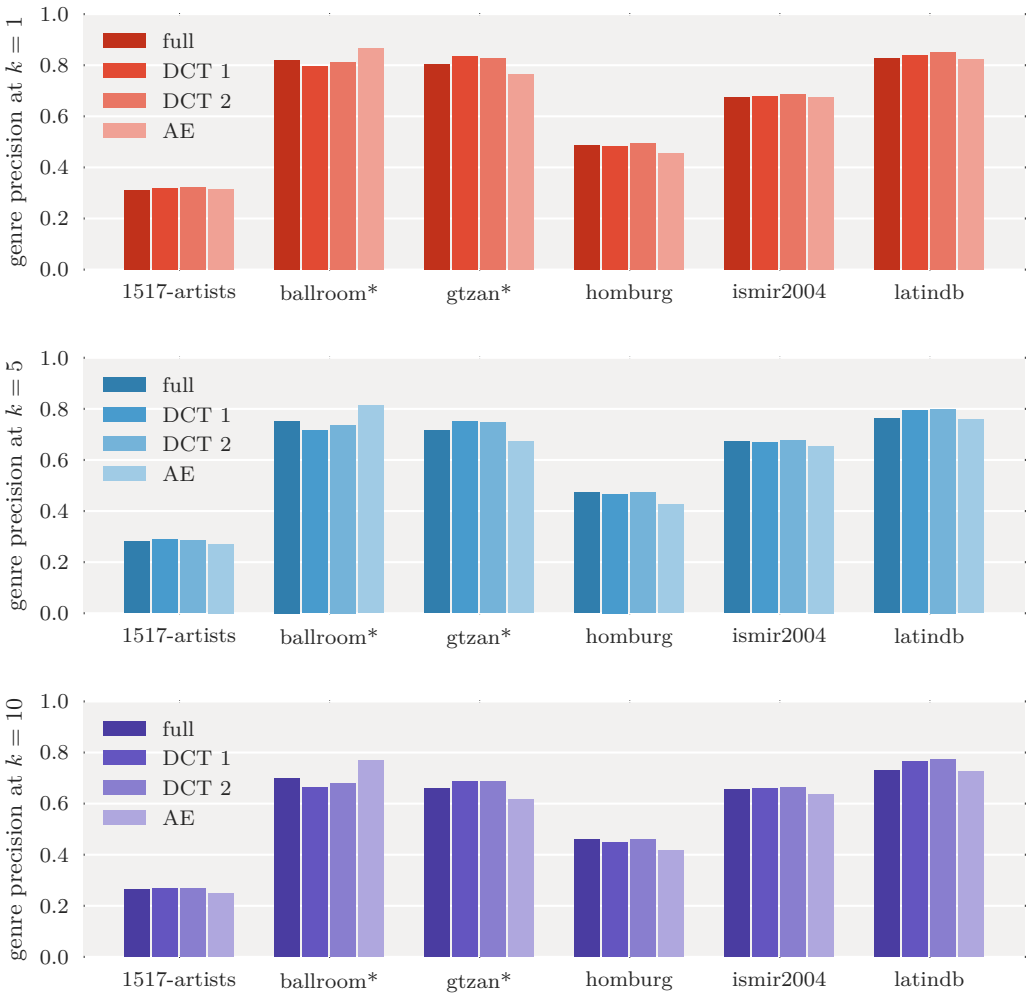


Figure 6.6: Genre precision at $k \in \{1, 5, 10\}$ for three compression schemes compared to the original vector-based similarity measure, on six datasets: 1517-artists (Seyerlehner et al., 2010b), ballroom (Gouyon et al., 2004), gtzan (Tzanetakis and Cook, 2002), homburg (Homburg et al., 2005), ismir2004 (`http://ismir2004.ismir.net/genre_contest/`), latindb (Silla Jr. et al., 2008). Those marked with an asterisk lack artist information and have been evaluated without an artist filter.

As results seemed promising, I put the AE-based similarity measure to a practice test by integrating it into the "Wolperdinger" prototype of Schnitzer (2011, Sec. 6.3), a browser-based music exploration interface for 2.3 million songs. Unfortunately, while it was blazingly fast, results were not qualitatively convincing compared to the similarity measure of Pohle et al. (2009), despite looking good on paper (i.e., Figure 6.6).

In an attempt to improve the measure, I fine-tuned the AE using HDML, this time building triplets (x, x^+, x^-) based on the items' genre information rather than their distance as measured by an existing similarity measure. However, this did not have any positive effect.

My conclusion is that current music similarity estimation methods have reached a level of quality at which genre labels are not a good enough proxy for comparing or improving their performance. Lacking an alternative, I ended this line of research.

6.8 Discussion

In this chapter, I have shown how to learn binary codes for song representations of two state-of-the-art music similarity measures that are useful for fast k-NN retrieval. While the techniques themselves were already known and successfully applied to documents and images, it was not clear whether they would apply to the highly non-metric spaces of the two music similarity measures. Furthermore, I demonstrated that for collection sizes encountered in music information retrieval, a k-NN index based on a linear scan can outperform sublinear-time methods when we require a particular accuracy – even more so as scan-based methods are *embarrassingly parallel* and can be easily distributed or performed on a GPU.

The scenario and experiments are based on some assumptions that are worth discussing:

(1) My experiments explicitly targeted commercial-scale collections and song-level search. For user collections, PCA or FastMap will be preferable as they can quickly adapt to any dataset; training AE and HDML took 20 and 180 minutes, respectively. For similarity search on a finer scale (e.g., 10-second snippets), the increased number of items may require sublinear-time methods.

(2) All timings and the results derived from it assume that the similarity models are kept in main memory. As noted in Section 6.7, for the vector-based measure, this requires about 35 GiB of memory per million songs. If this is infeasible, we could resort to high-speed Solid-State Drives (SSDs), slowing down the refine step by at least a factor of ten. As the prefiltering speed is not affected, this reduces ρ_t to one tenth, making methods with low ρ_s much more attractive: 1024-bit AE would give a 750x speedup over the (now much slower)

baseline, answering 1.5 1-NN queries per second. But even if we find a way to keep all models in memory, the refine step operates close to the bandwidth of 2013-era main memory: Computing 1 million 9,448-dimensional Manhattan distances in 2.361 s requires reading $10^6 \cdot 9448 \cdot 4\,\mathrm{B}/2.361\,\mathrm{s} \approx 15\,\mathrm{GiB/s}$. If we want to answer multiple queries in parallel, this would again slow down the refine step, and change the balance of ρ_t and ρ_s. My follow-up experiments in Section 6.7 indicate that the vector-based models can be compressed to 10% of their size with a minor impact on accuracy (also cf. Seyerlehner et al., 2010c). This solves the bandwidth problem, but again changes the balance, now in favour of faster and less accurate prefiltering.

(3) Methods are compared at the point at which they retrieve 90% of the true k nearest neighbours to a query. This was motivated by the fact that music similarity estimation is a very ill-defined problem: How can we even assume that music similarity is a one-dimensional phenomenon that can be expressed as a scalar, and is independent from a song's or a user's context? To avoid opening a can of worms, I opted to offload assessing the quality of results returned by a particular combination of filtering and refinement to the music similarity measure. Of course, there are different ways to do so. My choice of comparing the k retrieved items to the k true nearest neighbours means that any retrieved item that is not among the first k true neighbours is treated as a miss (no matter how near or far off it is). Instead, we could compute the sum of the true ranks of the k retrieved items, or assume all retrieved items within a particular similarity range of the query to be a relevant result. And finally, we could evaluate retrieved items in a qualitative way (as I tried in Section 6.7). All of this could drastically change the results and conclusions.

To improve results for the scenario investigated in this chapter, we could extend the approach to a hierarchical filter-and-refine: Use a very fast method as a first prefilter (possibly even a sublinear-time method with lower selectivity) and a more accurate method as a second prefilter, either to cut down the total prefilter time or to increase selectivity. A key idea of semantic hashing is to learn codes that are small enough to be used as memory addresses, to allow instant retrieval of items of the same code (Salakhutdinov and Hinton, 2009b) or similar codes (Norouzi et al., 2012b). In my experiments, such short codes were not competitive, but a two-stage prefilter could change this (cf. Krizhevsky and Hinton, 2011).

For the overarching task of music similarity estimation, in the light of the other chapters of this thesis, a promising route for future work would be to learn a similarity measure with a deep neural network trained on spectrograms, on existing high-level features, or even by formulating hand-designed similarity measures like Seyerlehner's as a neural network (i.e., as a function differentiable with respect to some parameters that might be worth optimizing). However, it is neither clear

what kind of data to train on, nor how to measure progress – both the ubiquitous genre classification and human judgements on a small-scale collection as used in MIREX seem to have reached their limits (cf. Flexer, 2014). Data derived from a large-scale collaborative filtering system might provide an alternative, but could be biased towards popular items (see Appendix A.3, p. 202), and is not easily available to researchers.

7 Musical Onset Detection

In this chapter, we return to applying deep learning to spectrograms, addressing the first event detection task of this thesis: Musical onset detection. It entails finding the starting points of all musically relevant events in an audio signal, such as notes being played or sung and drums being hit, without attempting to identify the instruments or pitches. While one of the most elementary tasks in music analysis, it is still only solved imperfectly for polyphonic music signals.

In particular, I explore using Convolutional Neural Networks (CNNs) for this task. Interpreting musical onset detection as a computer vision problem in spectrograms, CNNs seem to be an ideal fit for solving it. In experiments on a dataset of about 100 minutes of music with 26k annotated onsets, I show that CNNs outperform the previous state of the art while requiring less manual preprocessing of the input data. Investigating their inner workings, I find two key advantages over

hand-designed methods: Using separate detectors for percussive and harmonic onsets, and combining results from many minor variations of the same scheme. The results suggest that even for comparatively well-understood signal processing tasks, machine learning can be superior to knowledge engineering.

This work has previously been published in Schlüter and Böck (2013, 2014), in collaboration with Sebastian Böck, who provided his large onset detection dataset without which the experiments would not have been feasible, shared his evaluation code to enable fair comparison to his state-of-the-art results, and helped in writing the discussion of related work. The idea, conceptualization and realization of experiments as well as all other sections of the papers were exclusively my responsibility. This chapter roughly follows the second publication, with some content restructured or added, additional figures, and two new final sections.

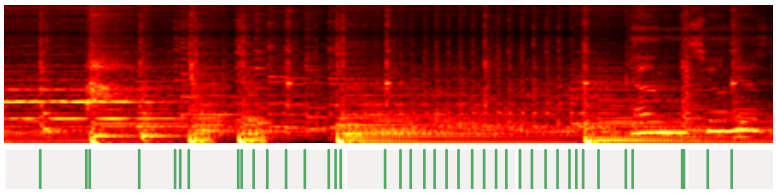
The remainder of this chapter is organized as follows: Following a more detailed introduction of the task and idea in Section 7.1 and a review of related work in Section 7.2, I will describe the network architecture and training method in Section 7.3, present quantitative experimental results in Section 7.4 and perform a qualitative analysis of a trained network in Section 7.5 to understand its inner workings. In Section 7.6, I briefly describe unpublished follow-up experiments, and round up with a discussion of results and insights in Section 7.7.

7.1 Introduction

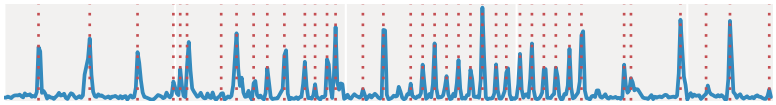
Detecting musical onsets can form the first step for several higher-level music analysis tasks such as beat detection, tempo estimation and transcription. On a spectrotemporal representation, onset detection is closely related to edge detection in images: Onsets are characterized by a swift change of spectral content over time, and can even be accompanied by wide-band transients clearly visible in a spectrogram (Figure 7.1a; see p. 50 for one reason the transients appear). Harmonic instruments without a strong attack (e.g., bowed instruments) lack transients, but manifest as thin lines over time with a beginning and end (Figure 7.1c).

Highlighting sharp oriented edges in an image requires local information only and can be accomplished by convolution with a small filter kernel, which is the basic operation of a CNN (see Equation 2.19, p. 19). In fact, even a random filter kernel will often act as an oriented edge detector. Figure 7.2 demonstrates this by convolving a greyscale photograph with a 5x5 patch of random values.

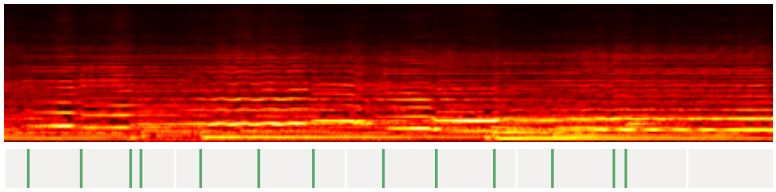
This lead to the idea of training such a network to find onsets in spectrogram excerpts: If a randomly initialized CNN already detects edges, it should quickly learn a set of suitable filter kernels to detect onsets. In this chapter, I will investigate how well this idea plays out in practice, and also shed some light on how the learned solutions work.



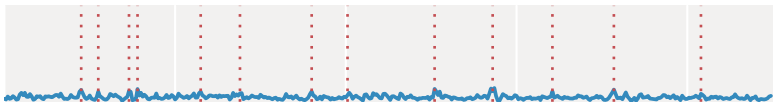
(a) mel spectrogram and annotated onsets for a Latin music excerpt



(b) activation curve and picked peaks of a hand-designed detector



(c) mel spectrogram and annotated onsets for a violin/piano excerpt



(d) activation curve and picked peaks of a hand-designed detector

Figure 7.1: Mel spectrogram and onset annotations for a 4.5-second recording rich in percussive onsets (a) and another one featuring smooth violin notes (c). A hand-designed method based on detecting spectral differences over time works well on the former (b), but not on the latter (d).

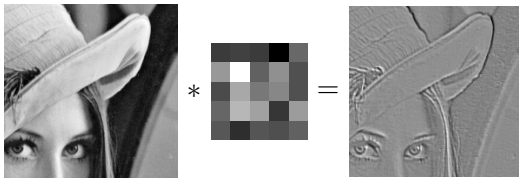


Figure 7.2: Convolving an image (*left*) with a random 5x5 kernel (*centre*, enlarged) can find oriented edges (*right*).

7.2 Related Work

First attempts at onset detection relied on traditional signal processing, exploiting changes of spectral energy (Bello et al., 2005; Dixon, 2006; Holzapfel et al., 2010; Böck and Widmer, 2013), pitch (Collins, 2005; Holzapfel et al., 2010) or phase (Bello et al., 2004; Dixon, 2006; Holzapfel et al., 2010) accompanying an onset. For example, Böck and Widmer (2013) compute a cent-scaled magnitude spectrogram, apply a sliding maximum filter over frequencies to blur vibrato, compute the elementwise difference between nearby spectral frames, and sum up all positive differences per time step to obtain an onset activation function. Peaks in this function (detected by comparing the maximum in a particular neighbourhood with the mean, to become independent of the input signal's local loudness) are then reported as onsets. The results of this method are shown in Figures 7.1b and 7.1d, with the activation function as a solid line, and detected peaks as dashed vertical lines: While it easily detects the transients of percussive onsets, it performs less well for soft violin notes.

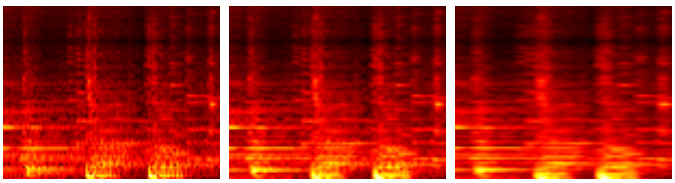
As an alternative to hand-design, several authors successfully explored neural networks for this task. Marolt et al. (2002) use neural networks to improve peak picking on a hand-crafted onset detection function, but do not learn the detection function itself and restrict their experiments to piano music. Lacoste and Eck (2006) learn an onset detector on spectral data with neural networks, but propose convolution for future work only. Eyben et al. (2010) train a bidirectional Recurrent Neural Network (RNN) on mel-scaled magnitude spectrograms preprocessed with a time difference filter. Böck et al. (2012) refine this model in several steps, defining the state of the art in onset detection when I worked on this task in 2013.[1]

Convolutional learning on music audio data has been evaluated for genre and artist classification (Lee et al., 2009b; Li et al., 2010; Dieleman et al., 2011), tagging (Hamel et al., 2011), key detection (Dieleman et al., 2011) and chord detection (Humphrey and Bello, 2012). Although results were promising, in 2013, CNNs had not yet been applied to the comparably low-level task of onset detection. Moreover, CNNs were treated as black boxes – while Lee et al. (2009b) visualized some of the learned features, they did not attempt to explain how the networks performed genre and artist classification.

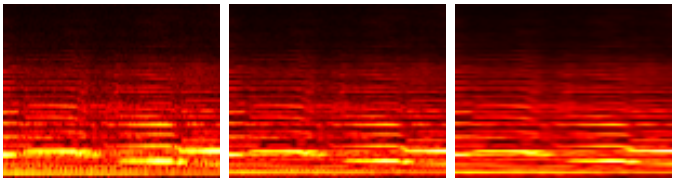
7.3 Method

After describing how the input data is preprocessed for the network, I will explain and give reasons for the chosen network architecture and finally discuss how I train the model.

[1] Also see results at `http://nema.lis.illinois.edu/nema_out/mirex2011/results/aod/` and `http://nema.lis.illinois.edu/nema_out/mirex2012/results/aod/`, accessed May 2017.



(a) Latin music excerpt



(b) violin/piano excerpt

Figure 7.3: Mel spectrograms for one-second excerpts of the recordings of Figure 7.1, computed with underlying STFT frame lengths of 1024, 2048 and 4096 samples, respectively. Shorter frames are blurry in pitch, longer frames are blurry in time.

7.3.1 Input Features

Following Böck et al. (2012), from the monophonic input signals, I compute three magnitude spectrograms with a hop size of 10 ms and frame lengths of 23 ms, 46 ms and 93 ms, respectively (i.e., 1024, 2048 and 4096 samples at sample rate 44.1 kHz, with the same hop size of 441 samples). Using multiple frame lengths elegantly works around the tradeoff between temporal resolution and frequential resolution explained on p. 51. To each of the three spectrograms, I apply an 80-band mel filterbank from 27.5 Hz to 16 kHz and scale magnitudes logarithmically with $\log(\max(x, 2.22\text{e}{-22}))$ as done by yaafe (Mathieu et al., 2010). This way, all mel spectrograms share the same frame rate and frequency mapping, but the effect of the different frame lengths is still clearly visible. Figure 7.3 demonstrates this for excerpts of the two examples of Figure 7.1: Short windows provide a clear localization of percussive onsets, but make it harder to distinguish different pitches of harmonic instruments, while long windows behave the opposite. As we see, both temporal and frequential resolution can be relevant for detecting onsets.

Finally, for easier digestion by the network, I normalize each frequency band to zero mean and unit variance computed over a hold-out dataset (i.e., using separate normalization constants per frame length and band, but not per file). Standardized features are assumed by most network parameter initialization schemes (Section 2.2.4.4, p. 33), and standardizing each frequency band separately achieves more similar feature distributions in the different bands, making the input more amenable for convolution over the frequency dimension.

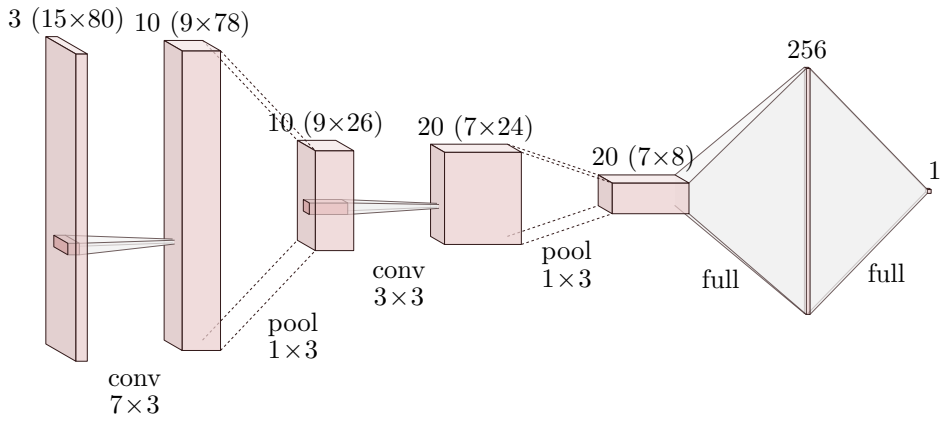


Figure 7.4: The Convolutional Neural Network architecture used in this work. Starting from a stack of three spectrogram excerpts, convolution and max-pooling in turns compute a set of 20 feature maps classified with a fully-connected network of 256 hidden units.

7.3.2 Network Architecture

To be used as an onset detector, I train a network on spectrogram excerpts centred on the frame to classify, giving binary labels to distinguish onsets from non-onsets. Similarly to music and speech detection in Chapter 5, this turns the problem into a binary classification problem: Given a short spectrogram excerpt, is there a note onset at the centre? Using a sigmoid output unit (as is common for binary classification, see p. 16 in Section 2.2.2), the network will return a value between 0.0 and 1.0 indicating the onset probability. Once trained, the network can thus be applied to maximally overlapping excerpts of a recording to obtain an onset activation function similar to the hand-designed one shown in Figure 7.1b. This function is smoothed by convolution with a Hamming window of 5 frames, and local maxima higher than a given threshold are reported as onsets.

As argued in Section 7.1, onset detection largely requires detecting specific oriented edges in a spectrogram. This is a local operation which can be solved with two-dimensional convolution. Thus, instead of fully-connected layers as in Chapter 5, I use 2D convolutional layers. As argued for on p. 21, I interleave convolutional layers with max-pooling layers, and end with fully-connected layers integrating the information from all input positions. In particular, I use two convolutional and pooling layers to perform local processing, a single fully-connected hidden layer and a single output unit, as shown in Figure 7.4. While this architecture is inspired from networks used for image classification such as LeNet (LeCun et al., 1998a) or AlexNet (Krizhevsky et al., 2012), there are some interesting differences.

For one, computer vision often handles colour images, presenting the input such that each application of a convolutional filter accesses the same local region in all colour channels (e.g., red, green, and blue). Here, the input consists of three mel spectrograms with different frame lengths, but the same frame rate, reduced to the same number of frequency bands with mel filter banks. I treat these as three input channels, so each output is computed from information of different temporal and frequential accuracies for its location in the time-frequency plane.

Secondly, computer vision usually uses square filters, and square pooling. In spectrograms, the two dimensions represent two different modalities, though. In preliminary experiments, I obtain improved results with rectangular shapes (cf. Humphrey and Bello, 2012). In particular, as the task mostly entails finding changes over time, I use filters wide in time and narrow in frequency, and as the task requires results of high time resolution, but is oblivious to frequency, I perform max-pooling over frequencies only.[2]

Note that we can exploit the convolutional architecture when detecting onsets in a test signal. Instead of explicitly applying the network to overlapping spectrogram excerpts, which would result in redundant computation in the first layers, we can apply the convolutional and pooling layers to a full-length spectrogram at once, and then feed overlapping excerpts of the feature maps to the fully-connected layer. Equivalently, we can cast the first fully-connected layer as a convolutional layer that spans the full frequency range, and the output unit as a 1D convolutional layer processing the hidden representations over time. See Appendix B.2 for details.

7.3.3 Training Methodology

The networks are trained using mini-batch gradient descent with momentum, minimizing the binary cross-entropy error. To improve generalization, I optionally apply 50% dropout to the inputs of the two fully-connected layers (see p. 38 in Section 2.2.5). As a second measure, I noted that for the given spectrogram frame rate (100 Hz), assigning each annotated onset to a single frame may be inappropriate – some annotations are not accurate enough, and some onsets are not that sharp –, so I optionally assign it to three frames instead, weighting the two extra frames less in training. That is, the cross-entropy loss for a single input example becomes

$$J(y, t, w) := w\big(-t \log(y) - (1 - t) \log(1 - y)\big), \tag{7.1}$$

where $t = 1$ if the input example has an annotated onset on or next to its central frame, $t = 0$ otherwise, and $w = 0.25$ if it has an onset next to the centre, $w = 1$ otherwise. I will quantify the effect of these two measures in the experiments.

[2] Incidentally, this type of max-pooling is also used in the hand-designed onset detector by Böck and Widmer (2013). Here it is part of the model, surrounded by automatically learned pre-processing and post-processing filters.

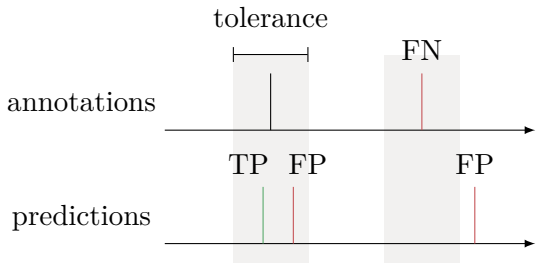


Figure 7.5: Illustration of the evaluation: Any predicted onset within a given temporal window around a yet unmatched ground truth annotation is a true positive (TP). Excess predictions are false positives (FP), and unmatched annotations are false negatives (FN).

7.4 Experimental Results

On a common dataset, I compare the CNN proposed in this work with three alternatives: A simple MLP as used as a baseline in Chapter 5, the hand-designed method of Böck and Widmer (2013) depicted in Figures 7.1b and 7.1d, and the RNN forming the state of the art at the time of these experiments. For the CNN, starting from an initial system, I perform several modifications to both architecture and training, yielding further improvements. I will report on these in detail after describing the dataset and evaluation methodology.

7.4.1 Dataset and Evaluation

All methods are evaluated on a dataset of about 102 minutes of music annotated with 25,927 onsets detailed by Böck et al. (2012, p. 4) and also used by Böck and Widmer (2013). It contains monophonic and polyphonic instrumental recordings as well as popular music excerpts.

As in Böck et al. (2012) and Böck and Widmer (2013), onset detection methods are compared by precision, recall and F-score. Specifically, a reported onset is considered correct (a true positive) if it is not farther than 25 ms from an unmatched target annotation; any excess detections and targets are false positives and negatives, respectively. See Figure 7.5 for an illustration. From these numbers, precision, recall and F-score are computed as in Section 5.4.4 (p. 86). To be comparable to Böck and Widmer (2013), for each method, I report results for the onset detection threshold yielding optimal F-score. As in Böck et al. (2012); Böck and Widmer (2013), all results are obtained in 8-fold cross-validation.

	Precision	Recall	F-score
MLP	0.843	0.815	0.828
RNN	0.892	0.855	0.873
CNN	0.905	0.866	0.885
+ Dropout	0.909	0.871	0.890
+ Fuzziness	0.914	0.885	0.899
+ ReLU	**0.917**	**0.889**	**0.903**
SuperFlux	0.883	0.793	0.836

Table 7.1: Performance of an MLP, the state-of-the-art RNN (Eyben et al., 2010, in its refined version by Böck and Widmer, 2013, Tab. 1), the proposed CNN and a hand-designed method (Böck and Widmer, 2013, Tab. 1). See Sections 7.4.2–7.4.5 for details on the CNN variants.

7.4.2 Initial Architecture

As described in Section 7.3.2, the CNN is trained on spectrogram excerpts centred on the frame to classify. Specifically, the network input for a single decision consists of the central frame plus a context of ±70 ms (15 frames in total), from all three spectrograms. While the RNN of Eyben et al. (2010) could, in theory, use a much longer context for its decisions, Sebastian Böck and me empirically found that after training it for onset detection, its output reaches a constant value within about 7 frames of an input change, so I chose a similar context size for the CNN.

The detailed architecture is depicted in Figure 7.4: From the 3-channel spectrogram excerpts of 15 frames by 80 bands, a convolutional layer with filters of 7 frames by 3 bands (by 3 channels) computes 10 feature maps of 9 frames by 78 bands. The next layer performs max-pooling over 3 adjacent bands without overlap, reducing the maps to 26 bands. Another convolutional layer of 3×3 filters and another 3-band max-pooling layer result in 20 maps of 7 frames by 8 bands (1120 neurons in total). These are processed by a fully-connected layer of 256 units and a final fully-connected layer of a single output unit predicting onsets. Both convolutional layers use the tanh nonlinearity (with a scalar bias per feature map), and the fully-connected layers use the logistic sigmoid.

The network is initialized following Equation 2.46 (p. 35) and trained in mini-batches of 256 examples, for 100 epochs, using a fixed learning rate of 0.05, and an initial momentum of 0.45, linearly increased to 0.9 between epochs 10 and 20.

It achieves an F-score of 88.5%, about one percent point above the state-of-the-art RNN and much better than a Multi-Layer Perceptron (MLP) of two hidden layers of 256 units trained in the same way as the CNN (Table 7.1, rows 1–3).

When trained on single-channel spectrograms instead, both the CNN and RNN lose about one percent point (not shown in table).

7.4.3 Bagging and Dropout

Bagging is a straightforward way to improve the performance of a classifier without changing its architecture: Training four RNNs and averaging their outputs gives a slight improvement to 87.7% F-score. Similarly, bagging two of the CNNs improves results to 89.1%, but four CNNs perform the same. A single CNN with twice the number of units in each layer overfits and obtains 87.9% only. Jointly training two CNNs sharing the same output unit does not overfit, but is inferior to bagging CNNs trained separately. I conclude that the benefit of bagging over simply enlarging the network does not stem from the fact that its constituent parts do not overfit, but that they are forced to solve the task on their own – when training two CNNs with a shared output unit, the second network will not receive any learning signal when the first produces the correct answer with high confidence and vice versa.

The same holds for the hidden units *within* each CNN. An elegant way to ensure that each unit receives a learning signal and is encouraged to solve its task independently of its peers is using dropout (p. 38 in Section 2.2.5): For each training case, half of the units are omitted from the network (cheaply accomplished by masking their output), chosen at random, and remaining weights are doubled to compensate. Applying this to the inputs of the two fully-connected layers and increasing the learning rate to 1.0, multiplied with 0.995 after each epoch,[3] yields 89.0% F-score. Note that dropout does not incur any higher costs at test time, while bagging two CNNs is twice as expensive. Another key advantage is that it limits overfitting, allowing us to fix training time to 300 epochs and try different setups without the need for early stopping on a validation set.

7.4.4 Fuzzier Training Examples

Onsets are annotated as time points. For training, I associate each annotation with its closest spectrogram frame and use this frame (along with its ±7 frames of context) as a positive example, and all others as negative examples. Some onsets have a soft attack, though, or are not annotated with 10 ms precision (the hop size of the spectrograms), resulting in actual onsets being presented to the network as negative training examples. To counter this, I would like to train on less sharply defined ground truth. One solution would be to replace the binary targets with sharp Gaussians and turn the classification problem into a regression one, but preliminary experiments on the RNN showed no improvement with this

[3] Hinton et al. (2012b, Sec. A.1) suggest to use a large initial learning rate with exponential decay rather than a constant learning rate when using dropout, and this also worked well here.

measure. Instead, I define a single frame before and after each annotated onset to be additional positive examples. To still teach the network about the most salient onset position, these examples are weighted with only 25% during training. This improves F-score to 89.9%, using a higher detection threshold than before, as the network output becomes larger on average. Simply excluding 1 or 2 frames around each onset from training, letting the network freely decide on those (i.e., setting $w = 0$ in Equation 7.1 for the frames next to an onset frame), works almost as well.

7.4.5 Rectified Linear Units

Both the hand-designed SuperFlux algorithm (Böck and Widmer, 2013) and the state-of-the-art RNN (Eyben et al., 2010) build on precomputed positive differences in spectral energy over time. Replacing the tanh activation function in the convolutional layers with the linear rectifier $y(x) = \max(0, x)$ (see p. 17) provides a direct way for the CNN to learn to compute positive differences in its spectral input, and has been shown generally useful for supervisedly trained networks (Glorot et al., 2011). In this case, it improves F-score to the final result of 90.3%. For some reason, using rectified linear units for the fully-connected hidden layer as well reduces performance to 89.6%.

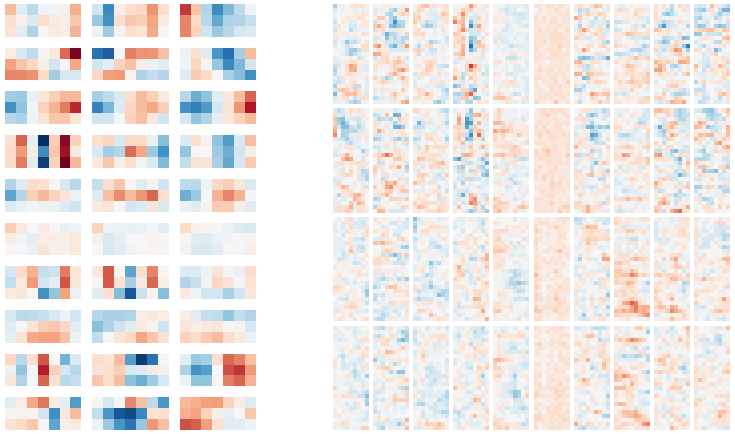
7.5 Network Examination

While I obtained a state-of-the-art musical onset detector that is perfectly usable as a black box, I would like to know how it works. In particular, I hope to gain some insights on why it is better than existing hand-crafted algorithms, and possibly learn from its solution to improve these.

For this purpose, I train a simplified CNN with the second convolutional layer and max-pooling layer removed to make it easier to interpret, and tanh units for the remaining convolutional layer. It achieves 88.8% F-score, which is still far superior to the hand-designed SuperFlux algorithm (Table 7.1, last row), making it an interesting model to study. I will visualize both the connections learned by the model and its hidden unit states on test data to understand its computations.

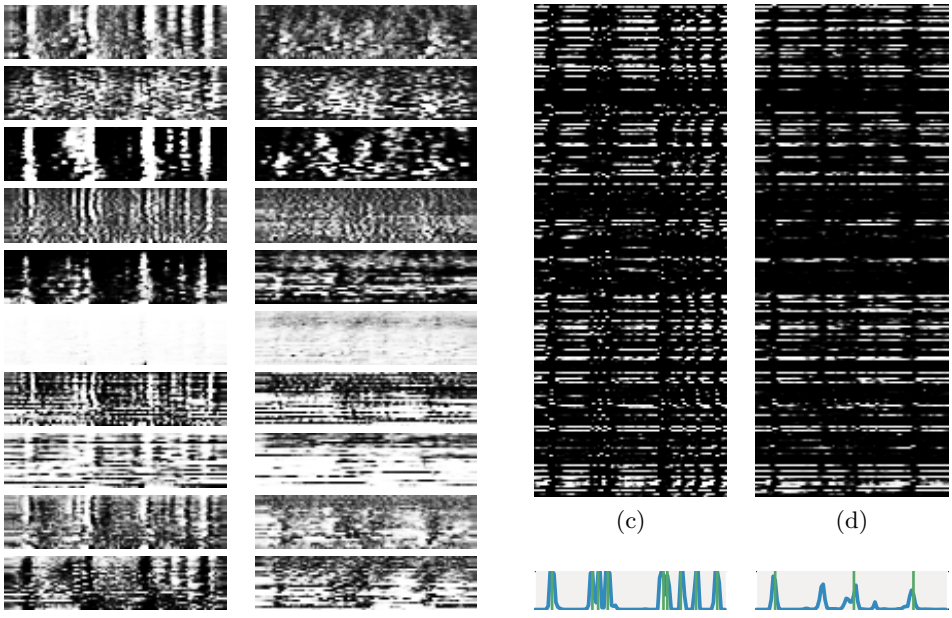
7.5.1 Learned Filters

In a first attempt to understand the network, I visualize its filters after training. Figure 7.6a depicts the convolutional filters of the first layer processing the 3 input spectrograms of different frame lengths. The third row shows a filter that clearly computes a difference over time with some blurring of frequencies in all three inputs, the filter in the first row computes a difference between two of the spectrograms, other filters seem to compute very different things for the different input channels.



(a) all 10 first-layer filters (b) 4 of the 256 second-layer weights

Figure 7.6: Visualization of the weights learned by the simplified CNN. Each block depicts the filter connecting a particular input channel (in columns) with a particular output (in rows), with red and blue denoting negative and positive values, respectively. The first layer has 10 convolutional units with 7×3 filters applied to the three spectrogram channels (of frame lengths 2048, 1024, 4096). The second layer has 256 fully-connected units processing 10 feature maps of size 9×26; only 4 units shown here.



(a) (b) (c) (d) (e) (f)

Figure 7.7: Visualization of the two excerpts of Figure 7.3 passing through the CNN: The 10 feature maps after max-pooling and tanh activation (a, b), the 256 activations of the fully-connected layer (c, d), and the final output with ground truth marked by vertical bars (e, f).

Figure 7.6b shows the weights of four of the 256 fully-connected units processing the 10 max-pooled feature maps of the first layer. Since I cannot show all 256 units, I selected the two units with the largest positive and negative connection weights to the output unit, respectively, assuming that these are the most interesting. However, they are hardly interpretable.

7.5.2 Data Transformation

In a second attempt, I visualize the data propagated through the network, for the two music excerpts already shown in Figure 7.3. The excerpts are well-chosen, with one rich in percussive onsets, the other in transient-free harmonic ones.[4] Figures 7.7a and 7.7b show the feature maps produced by the 10 filters of Figure 7.6a after max-pooling and the nonlinear tanh activation function.[5] The percussive onsets of the first example are clearly visible in most of the feature maps, as white vertical structures (i.e., positive values) preceded or followed by black vertical structures (i.e., negative values) – however, the onsets are clearly visible in the original spectrogram as well (Figure 7.3a). The three rapidly succeeding onsets about one third into the excerpt are sometimes blurred into one. The harmonic onsets of the second example appear as faint light (first feature map) or dark (last two feature maps) structures. The activations of the 256 fully-connected units in Figures 7.7c and 7.7d also reflect the onsets, mostly as dark vertical lines (i.e., many units are inactive at onsets). While it is conceivable how this could lead to the final output (Figures 7.7e and 7.7f), it is still hard to interpret.

7.5.3 Backtracking

So far, we only understood that the network appears to compute differences over time, but it is unclear how this maps to the final output. A problem is that even in this comparatively small and shallow CNN, there are many filters and activations to look at. As a third attempt, we will start at the output unit and work our way backwards through the network, concentrating on the parts that contribute most to its classification decisions.

Output unit: The output unit computes a weighted sum of the 256 hidden unit states below, then applies the logistic sigmoid function, resulting in a value between 0.0 and 1.0 interpretable as an onset probability. Figures 7.7e and 7.7f show this output over time for the two test signals. Except for a false positive in the latter, the network output well matches the ground truth (denoted by vertical lines). To understand how the output is driven by the

[4] Audio available at http://jan-schlueter.de/pubs/2014_icassp/, accessed May 2017.
[5] Usually the nonlinearity comes first, but since tanh is monotonic, we can change the order.

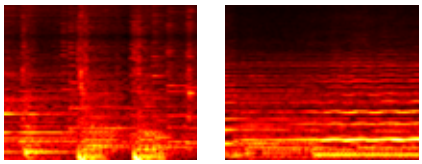
256 hidden units, I visualize their states for the two signals as in Figures 7.7c and 7.7d, but this time ordered by connection weight to the output unit (Figure 7.8c). Interestingly, the most strongly connected units (near the top and bottom border) are hardly active and do not seem to be useful for these examples – they may have specialized to exotic corner cases in the training data (the weights of these units were shown in Figure 7.6b). In contrast, a large number of units with small connection weights (near the sign change prominently visible in Figure 7.8c) clearly reflects the onset locations, either by activating or deactivating at onsets. Comparing states for the two signals, we see that a number of positively connected units (below the sign change) detect percussive onsets only, while others also detect harmonic ones.

Fully-connected hidden layer: Having identified the most interesting hidden units (the ones near the sign change), we will investigate what they compute. Figure 7.8d visualizes the connections of four units to the feature maps in the layer below. The last one displays a sharp wide-band off-on-off connection to the fourth map (last row, fourth column), and similarly sharp connections to other maps. It is good in detecting percussive onsets, which are short wide-band bursts. The second unit computes more long-term differences, notably in the first and ninth map, and manages to capture harmonic onsets. Other units look very similar to the types shown, with variations in timing and covered frequency bands.

Convolutional layer: To close the remaining gap to the input, we will review the feature maps computed by the convolutional layer. From the previous investigation, maps 4 and 9 seem to play an important role. For the first signal, map 4 highlights the onsets very sharply (Figure 7.8e). Looking at the corresponding filter (Figure 7.8g), it seems to detect energy bursts of 1 to 3 frames in the spectrogram of mid-sized frame length, and compute a temporal difference in the one of longer frame length. Map 9 also computes this temporal difference and contrasts it against a slightly offset difference in the spectrogram of short frame length (Figure 7.8h). While still very fuzzy, this enhances onsets of the second signal (Figure 7.8f).

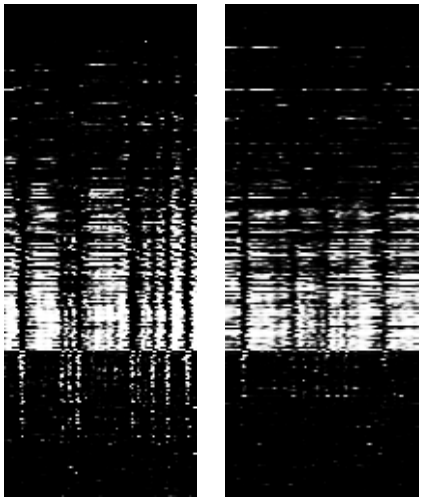
7.5.4 Insights

Although our inspection was highly selective, covering a small part of the network only, we formed a basic intuition of what it does. Like spectral flux based methods, the network computes spectral differences over time. In doing so, it adapts the context to the spectrogram window length, which was also found to be crucial by Böck and Widmer (2013). And like Holzapfel et al. (2010), the CNN separates the detection of percussive and pitched onsets. As a novel feature, the network



(a) input spectrograms

(b) network output (blue line) and ground truth (vertical green bars)

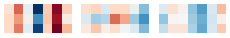


(c) penultimate layer states ordered by connection weight to output, from strongly negative (top) to strongly positive (bottom)

(d) weights of four penultimate layer units near the sign change from small negative to small positive connection weights to the output unit

(e) feature map 4 for first excerpt

(f) feature map 9 for second excerpt



(g) filters for map 4

(h) filters for map 9

Figure 7.8: Selected network weights and states for the two excerpts of Figure 7.3. The output (b) is a weighted combination of the hidden unit states (c), the most interesting of which seem to be those with small connection weights to the output. The filters of those units (d) prominently make use of feature maps 4 and 9 (e, f), computed from the inputs (a; only one of three channels shown) using convolutional filters (g, h).

computes the difference of short- and long-window spectrograms to find onsets. However, imitating this is not enough to build a good onset detector. In fact, judging by the states and filters of the 256 hidden units, the key factor seems to be that the network combines hundreds of minor variations of the same approach, something that cannot be reproduced with hand-designed algorithms.

7.6 Extensions and Dead Ends

For an independent evaluation of the method, I submitted it to the Music Information Retrieval Evaluation eXchange (MIREX), an annual facility for researchers to compare their solutions to different music analysis tasks (Downie et al., 2010). It scored better than all other approaches[6] and has not been surpassed since.[7]

After this initial success, I tried a few ideas that could have improved results further, but did not turn out to, as described in the following.

As noted in the discussion of related work (Section 7.2, p. 122), several hand-designed methods include the phase information in addition to or instead of the magnitudes – phase discontinuities can indicate onsets complementarily to magnitude differences. Similarly to how the CNN in this work already processes multiple magnitude spectrograms computed with different frame lengths, it could simultaneously process the phases and magnitudes or the real and imaginary part of a complex spectrogram in two channels. I experimented with both variants, applying the mel filterbank to the complex spectrogram before splitting it into magnitude and phase, or omitting the mel filterbank altogether. Neither improved results. This could either be due to the dataset – maybe it does feature too few onsets better detectable by phase information to make a difference in evaluation, or to give suitable hints in training – or because I have not found the best way to present the phase information to the network.

In an attempt to simplify the task for the network, I tried preprocessing the magnitude spectrograms with a Harmonic/Percussive Source Separation (HPSS). In the form proposed by FitzGerald (2010), it uses oriented median filters to separate a spectrogram into two parts that feature mostly harmonic and mostly percussive sources, respectively, in a way that an elementwise addition of the two parts reconstructs the original spectrogram. Again, a CNN can process these parts as separate input channels, with the interesting side aspect that it can simply access the unseparated magnitudes with filters that are identical for the two channels (recall that a convolutional unit processes each channel with a separate filter, then adds up the results, Equation 2.18, p. 19). Again, this did not improve results, but even deteriorated them. Possibly it would require the network to be initialized with sets of identical filters, so it initially sees the original spectrograms, or the still imperfect separation is simply not suited better for the task of onset detection than what the CNN can do on its own.

Finally, the context of the CNN used for a single detection is very limited. While it is probably large enough for otherwise uninformed detection of short-term events

[6] `http://nema.lis.illinois.edu/nema_out/mirex2013/results/aod/`, accessed May 2017

[7] Except by an integration of my MIREX submission into the `madmom` library by Böck et al. (2016), which performed marginally better in MIREX 2016 due to implementation differences: `http://nema.lis.illinois.edu/nema_out/mirex2016/results/aod/`, accessed May 2017

(merely doubling it does not improve results), it is conceivable that human listeners exploit the repetitiveness of music to anticipate onsets at certain times, and can therefore perceive onsets difficult to detect otherwise. In a simple attempt to enable the model to make use of more context, I extended it into a recurrent convolutional neural network, by replacing the fully-connected layer or layers with recurrent ones. A similar architecture – convolutional preprocessing followed by recurrent units – is also used in state-of-the-art speech recognition systems (Amodei et al., 2015). However, this slowed down training considerably, and I was not able to iterate fast enough to even just match the performance of the basic CNN.

7.7 Discussion

Through a combination of neural network training methods that were fairly recent in 2013, I significantly advanced the state of the art in musical onset detection. Analysing the learned model, I found that it rediscovered several ideas used in hand-designed methods, but is superior by combining results of many slightly different detectors. This shows that even for easily understandable problems, labelling data and applying machine learning may be more worthwhile than directly engineering a solution. Indeed, the motivation for combining this method and task was twofold: On the one hand, CNNs seemed an ideal fit for the task, possibly improving over existing solutions, and on the other hand, onset detection seemed an ideal task for exploring the internals of a neural network processing audio signals, given its simplicity in terms of required processing power, abstraction and temporal context.

Further small improvements of the method might be achievable by modifying the input representation or network architecture, but a more promising avenue would be increasing the size of the dataset – a key factor for the success of Sebastian Böck's and my submissions to MIREX is the large set of training data. Further insights into the inner workings might be won by modern CNN visualization techniques (Zeiler and Fergus, 2014; Simonyan et al., 2014; Springenberg et al., 2015), which employ a more principled way of back-tracking a classification decision through the network than my approach in Section 7.5.3, and which I will use in Chapter 9.

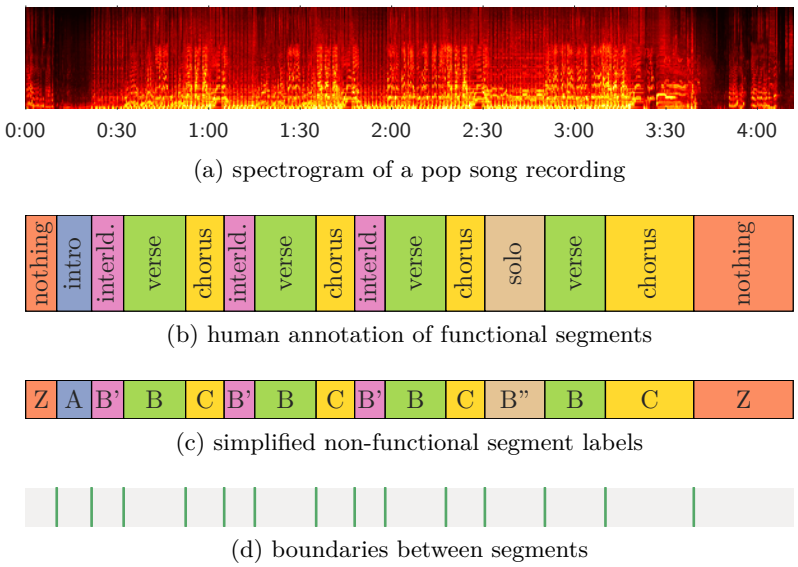
A potential disadvantage of the CNN proposed in this work over the RNN of Eyben et al. (2010) – a rather small network of three layers of 20 bidirectional Long Short-Term Memory (LSTM) units each – is its higher computational complexity for computing predictions (although the CNN is better parallelizable since it does not require processing an input in sequence, so it can be faster in practice depending on available hardware). It would be interesting to evaluate how a smaller CNN of matching complexity performs compared to the RNN. Another direction would be to integrate ideas from CNNs into RNNs, such as local connectivity and pooling, in order to improve their performance without increasing computational costs.

A second potential disadvantage of the CNN compared to an RNN is its fixed temporal context for a decision. While an RNN can, in theory, retain information indefinitely while processing a temporal sequence and use it when needed, the context of a CNN is predefined (e.g., the model of Figure 7.4, p. 124 always uses 15 spectrogram frames for a decision). On the other hand, onset detection may neither require a variable-length context, nor a wider context than used in this work – while in theory, this would allow to exploit the temporal structure of music to anticipate onsets, this will be difficult to realize in practice, since the majority of onsets in the training data cannot be accurately predicted from past onsets anyway (see Figure 7.1a, p. 121). Under this assumption, the fixed temporal context of a CNN poses an advantage: It allows to compute predictions at any position in an audio signal without processing all of the signal preceding it, enabling better parallelization and thus faster training.

8 Music Boundary Detection

We will now address a task on a higher level of abstraction: Recognizing structural boundaries in a music piece, i.e., finding the positions where the music changes in some key aspects such as rhythm, instrumentation or melody. In a pop music recording, an example would be the transition from verse to chorus, or from chorus to break. The goal is to automatically detect such boundaries in audio signals so that the results are close to human annotation.

Surprisingly, this can be treated as an event detection task and solved with the same approach I used in Chapter 7, adapting it to cater for the largely different time scale of structural boundary detection compared to onset detection. On a representative subset of the public SALAMI dataset (Smith et al., 2011), a CNN trained on mel spectrograms decisively outperforms previous approaches in terms of boundary retrieval F-score at two commonly evaluated temporal tolerances: It advances the state of the art from 0.33 to 0.46 for tolerances of ±0.5 seconds, and from 0.52 to 0.62 for tolerances of ±3 seconds. Moreover, since it is trained directly



(a) spectrogram of a pop song recording

(b) human annotation of functional segments

(c) simplified non-functional segment labels

(d) boundaries between segments

Figure 8.1: Structural segmentation of a music recording (a) entails determining and localizing its functional parts such as the chorus or verse (b), or, in a simplified form, localizing segments and identifying which ones are the same (c). Here, we consider the localization of segment boundaries only, not labelling the segments (d).

on annotated audio data and without task-specific expert knowledge, it should be easily adaptable to changed annotation guidelines or to related tasks such as the detection of song transitions.

This work was a close collaboration with Karen Ullrich and Thomas Grill, published in Ullrich et al. (2014), all three of us contributing about equally: Thomas and me proposed the task and dataset, Karen chose to try CNNs, Karen and me ran the experiments based on my onset detection implementation, Thomas oversaw the evaluation and established the lower and upper bound, and Thomas and me wrote the paper (as Karen was unavailable the week before the submission deadline). This chapter mostly follows the publication, with slightly extended descriptions, additional figures, and three new final sections. To reflect our collaboration, I will use first-person plural pronouns for all parts of the chapter I have not done alone.

The remainder of this chapter is organized as follows: Section 8.1 introduces the task and our idea in more detail, Section 8.2 gives an overview over related work, and Section 8.3 describes our proposed method. After presenting our main results in Section 8.4, in Section 8.5, I give some previously unpublished insights on how the network works. Section 8.6 describes follow-up work, and Section 8.7 wraps up with a discussion and outlook.

8.1 Introduction

The determination of the overall structure of a piece of audio, often referred to as its *musical form*, is one of the key tasks in music analysis. Knowledge of the musical structure enables a variety of real-world applications, be they commercially applicable, such as for browsing music, or educational. A large number of different techniques for automatic structure discovery have been developed, see Paulus et al. (2010) for an overview. In this work, we consider the subtask of retrieving the boundaries between the main structural parts of a piece of music. Figure 8.1 illustrates this task for an example recording. Finding the structural boundaries is a difficult task in itself, and can serve as an intermediate step in determining the musical form: Once the boundaries are known, the labels can be determined by comparing and grouping the segments (Paulus et al., 2010, Sec. 2).

Depending on the music under examination, the task of finding musical boundaries can be relatively simple or very difficult, leaving ample space for ambiguity in the latter case. In fact, two human annotators hardly ever annotate boundaries at the exact same positions. Instead of trying to design an algorithm by hand that works well in all circumstances, we let a CNN learn to detect boundaries from a large corpus of human-annotated examples.

8.2 Related Work

In the overview paper to audio structure analysis by Paulus et al. (2010), three fundamental approaches to segmentation are distinguished: Novelty-based, detecting transitions between contrasting parts, homogeneity-based, identifying sections that are consistent with respect to their musical properties, and repetition-based, building on the determination of recurring patterns. Many segmentation algorithms follow mixed strategies. Novelty is typically computed using Self-Similarity Matrices (SSMs) or Self-Distance Matrices (SDMs) with a sliding checkerboard kernel (Foote, 2000), building on audio descriptors like timbre (MFCC features), pitch, chroma vectors and rhythmic features (Paulus and Klapuri, 2008). Alternative approaches calculate difference features on more complex audio feature sets (Turnbull et al., 2007). In order to achieve a higher temporal accuracy in rhythmic music, audio features can be accumulated beat-synchronously. Techniques capitalizing on homogeneity use clustering (Foote and Cooper, 2003) or state-modelling (HMM) approaches (Aucouturier and Sandler, 2001), or both (Logan and Chu, 2000; Levy and Sandler, 2008). Repeating pattern discovery is performed on SSMs or SDMs (Lu et al., 2004), and often combined with other approaches (Paulus and Klapuri, 2006, 2009; McFee and Ellis, 2014). Some algorithms combine all three basic approaches (Serra et al., 2012).

Almost all existing algorithms are hand-designed from end to end. To the best of our knowledge, only two methods are partly learning from human annotations: Turnbull et al. (2007) compute temporal differences at three time scales over a set of standard audio features including chromagrams, MFCCs, and fluctuation patterns. Training Boosted Decision Stumps to classify the resulting vectors into boundaries and non-boundaries, they achieved significant gains over a hand-crafted boundary detector using the same features, evaluated on a set of 100 pop songs. McFee and Ellis (2014) employ Ordinal Linear Discriminant Analysis to learn a linear transform of beat-aligned audio features (including MFCCs and chroma) that minimizes the variance within a human-annotated segment while maximizing the distance across segments. Combined with a repetition feature, their method defined the state of the art in boundary retrieval when we did our experiments in early 2014. However, it still involves significant manual engineering.

For other tasks in the field of music information retrieval, supervised learning with CNNs has already proven to outperform hand-designed algorithms, sometimes by a large margin (Li et al., 2010; Dieleman et al., 2011; Hamel et al., 2011; Humphrey and Bello, 2012; Schlüter and Böck, 2014). In this work, we investigate whether CNNs are effective for structural boundary detection as well.

8.3 Method

We propose to train a neural network on human annotations to predict likely musical boundary locations in audio data. Our method is derived from Schlüter and Böck (2014) (described in Chapter 7), who use CNNs for onset detection: We also train a CNN as a binary classifier on spectrogram excerpts, but we adapt their method to include a larger input context and respect the higher inaccuracy and scarcity of segment boundary annotations compared to onset annotations. In the following, we will describe the features, neural network, supervised training procedure and the post-processing of the network output to obtain boundary predictions.

8.3.1 Input Features

For each audio file, we compute a magnitude spectrogram with a window size of 46 ms (2048 samples at 44.1 kHz) and 50% overlap, apply a mel filterbank of 80 triangular filters from 80 Hz to 16 kHz and scale magnitudes logarithmically. To be able to train and predict on spectrogram excerpts near the beginning and end of a file, we pad the spectrogram with pink noise at -70 dB as needed (padding with silence is impossible with logarithmic magnitudes, and white noise is too different from the existing background noise in natural recordings). To bring the input values to a range suitable for neural networks, we follow Schlüter and Böck (2014) in normalizing each frequency band to zero mean and unit variance. Finally, to

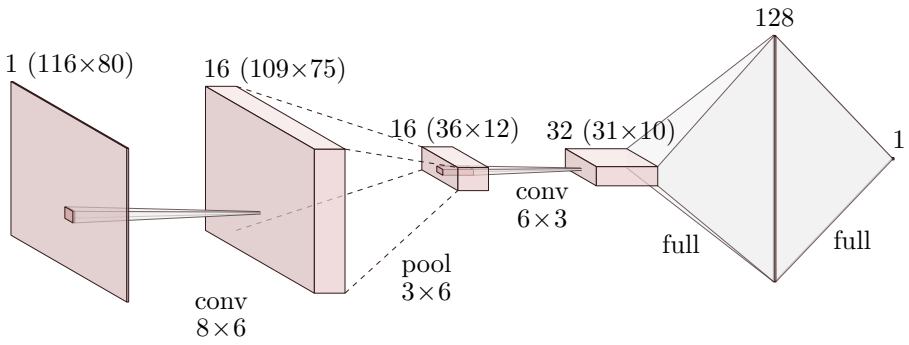


Figure 8.2: One of the Convolutional Neural Network architectures used in this work. Starting from a spectrogram excerpt, convolution and max-pooling in turns compute a set of 32 feature maps classified with a fully-connected network of 128 hidden units. The size of the feature maps depends on the input length, which we varied in $\{27, 58, 116\}$.

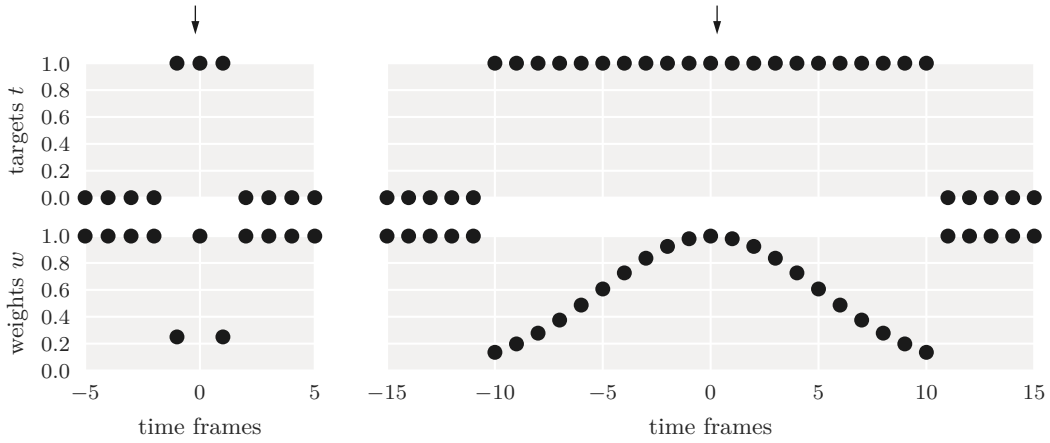
allow the CNN to process larger temporal contexts while keeping the input size reasonable, we subsample the spectrogram by taking the maximum over 3, 6 or 12 adjacent time frames (without overlap), resulting in a frame rate of 14.35 Hz, 7.18 Hz or 3.59 Hz, respectively.[1] We will refer to these frame rates as `high`, `std` and `low`.

We also tried training on MFCCs and chroma vectors (convolving over time only, as these descriptors have no or less continuity in the "vertical" feature dimension), as well as fluctuation patterns and self-similarity matrices derived from those. Overall, mel spectrograms proved the most suitable for the algorithm and performed best.

8.3.2 Network Architecture

Similar to Chapter 7, the network has to predict if a given spectrogram excerpt has a boundary at its centre. Again, we chose to process the input with two-dimensional convolutions. Specifically, as visualized in Figure 8.2, we fix the network architecture to a convolutional layer of 16 8×6 kernels (8 time frames, 6 mel bands, 16 output channels), a max-pooling layer of 3×6 (reducing inputs by a factor of 3 in time and of 6 in frequency), another convolution of 32 6×3 kernels, a fully-connected layer of 128 units and a fully-connected output layer of 1 unit. The convolutional layers use the tanh transfer function, the fully-connected layers use the sigmoid. This architecture was determined in preliminary experiments and not further optimized.

[1] Subsampling by maximum was chosen for convenience, as it allowed us to precompute the spectrograms and use a max-pooling network layer to tune the temporal resolution. We later compared this to mean-pooling without observing a significant difference.



(a) onset detection

(b) music boundary detection

Figure 8.3: For onset detection (a), I presented the three closest frames to an annotation (small arrow) as positive examples (upper panel), weighting the central one with 100% and the others with 25% in training (lower panel). Here, we extend this idea towards a longer time scale (b): Targets are positive within a given vicinity of an annotated segment boundary, and zero elsewhere (upper panel). Positive targets far from the annotation are given a lower weight, following a Gaussian function (lower panel).

8.3.3 Training Methodology

The input to the CNN is a spectrogram excerpt of N frames, and its output is a single value giving the probability of a boundary in the centre of the input. The network is trained in a supervised way on pairs of spectrogram excerpts and binary labels. To account for the inaccuracy of the ground truth boundary annotations (as observable from the disagreement between two humans annotating the same piece), we employ what we will refer to as *target smearing*: All excerpts centred on a frame within $\pm E$ frames from an annotated boundary will be presented to the network as positive examples, weighted by $w = \exp(-2d^2/E^2)$, where d is the distance to the annotated boundary in frames, and w scales the loss for the example (Equation 7.1, p. 125). Figure 8.3 illustrates this for $E = 10$, compared to the more short-term smoothing used for onsets in Chapter 7. Target smearing changes the binary classification task from "Is there a boundary at the centre of the excerpt?" to "Is there a boundary *near* the centre of the excerpt?". We will vary both the spectrogram length N and smearing environment E in our experiments. To compensate for the still severe scarceness of positive examples, we increase their chances of being randomly selected for a training step by a factor of 3.

Networks are initialized following Equation 2.46 (p. 35) and trained using stochastic gradient descent on cross-entropy error with mini-batches of 64 examples, momentum of 0.95, and an initial learning rate of 0.03 multiplied by 0.85 after every mini-epoch of 2000 weight updates. We apply 50% dropout to the inputs of both fully-connected layers (see p. 38 in Section 2.2.5). Training is always stopped after 20 mini-epochs. The validation error turned out not to be robust enough for early stopping (p. 37). Implemented in Theano (Al-Rfou et al., 2016), training a single CNN on an Nvidia GTX 780 Ti graphics card takes 50–90 minutes depending on the spectrogram length.

8.3.4 Postprocessing

At test time, we apply the trained network to each position in the spectrogram of the music piece to be segmented, obtaining a boundary probability for each frame.[2] We then employ a simple means of peak-picking on this boundary activation curve: Every output value that is not surpassed within ±6 seconds is a boundary candidate. From each candidate value we subtract the average of the activation curve in the past 12 and future 6 seconds, to compensate for long-term trends. We end up with a list of boundary candidates along with strength values that can be thresholded at will to obtain binary decisions. This scheme is equivalent to the first two conditions of the peak-picking method of Dixon (2006, Sec. 2.6). We found that more elaborate methods did not improve results.

8.4 Experimental Results

We perform a range of experiments first comparing different variants of our proposed methods against each other on a validation set, and finally evaluating the best two variants against the full range of existing methods submitted to the Music Information Retrieval Evaluation eXchange (MIREX, see Downie et al., 2010), as well as a lower bound and upper bound we derived for the data.

8.4.1 Dataset

We evaluate our algorithm on a subset of the Structural Analysis of Large Amounts of Music Information (SALAMI) database curated by Smith et al. (2011). In total, this dataset contains over 2400 structural annotations of nearly 1400 musical recordings of different genres and origins. About half of the annotations (779 recordings, 498 of which are doubly-annotated) are publicly available.[3] 1000 recordings of this

[2] See Appendix B, p. 205 on how this can be done efficiently despite temporal max-pooling.

[3] `http://github.com/DDMAL/salami-data-public`, accessed May 2017. We used version 1.2 of the annotations, the most current at the time of our work.

dataset were also used in the "Audio Structural Segmentation" task of the annual MIREX campaign. Along with quantitative evaluation results, the organizers published the ground truth and boundary predictions of all submitted algorithms for each recording. By matching the ground truth to the public SALAMI annotations, we were able to identify 487 recordings from the public SALAMI dataset. For these, we thus have access to the audio data, ground truth and the predictions of several existing methods. These serve as a test set, allowing us to evaluate our algorithm against the MIREX submissions from 2012 to 2014. We had another 733 recordings at our disposal, annotated following the SALAMI guidelines, which we split into 633 items for training and 100 for validation.

8.4.2 Evaluation

For boundary retrieval, the MIREX campaign uses two evaluation measures: *Median deviation* and *Hit rate*. The former measures the median distance between each annotated boundary and its closest predicted boundary or vice versa. The latter checks which predicted boundaries fall close enough to an unmatched annotated boundary (true positives), records remaining unmatched predictions and annotations as false positives and negatives, respectively, then computes the precision, recall and F-score (exactly as done for evaluating onset detection, see Section 7.4.1 and Figure 7.5 on p. 126). Since not only the temporal distance of predictions, but also the figures of precision and recall are of interest, we opted for the Hit rate as our central measure of evaluation, computed at a temporal tolerance of ± 0.5 seconds (as in Turnbull et al., 2007) and ± 3 seconds (as in Levy and Sandler, 2008). For accumulation over multiple recordings, we follow the MIREX evaluation by calculating F-score, precision and recall per item and averaging the three measures over the items for the final result. Note that the averaged F-score is not necessarily equal to the harmonic mean of the averaged precision and recall. Our evaluation code is publicly available for download.[4]

8.4.3 Baseline and Upper Bound

Our focus for evaluation lies primarily on the F-score. Theoretically, the F-score is bounded by $F \in [0, 1]$, but for the given task, we can derive more useful lower and upper bounds to compare our results to. As a baseline, we use regularly spaced boundary predictions starting at time 0. Choosing an optimal spacing, we obtain an F-score of $F_{\text{inf},3} \approx 0.33$ for ± 3 seconds tolerance, and $F_{\text{inf},0.5} \approx 0.13$ for a tolerance of ± 0.5 seconds. Note that it is crucial to place the first boundary at 0 seconds, where a large fraction of the music pieces has annotated segment boundaries. Many pieces have only few boundaries at all, thus the impact can be considerable. An upper

[4] http://jan-schlueter.de/pubs/2014_ismir/, accessed May 2017

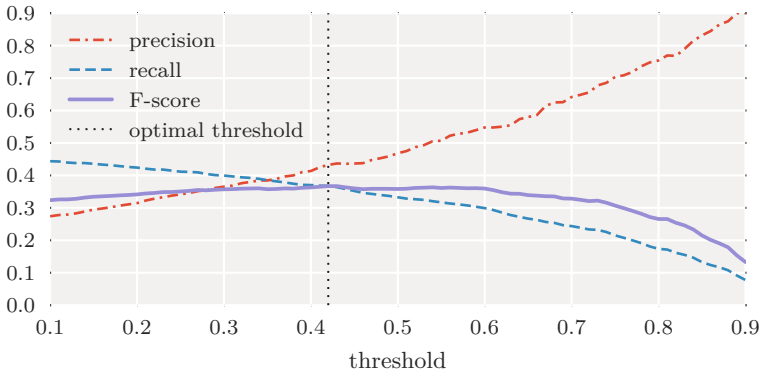


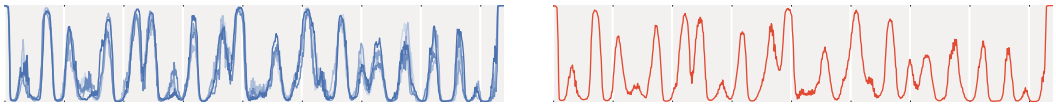
Figure 8.4: Optimization of the threshold shown for model `8s_std_3s` at tolerance ±0.5 seconds. Boundary retrieval precision, recall and F-score are each separately averaged over the 100 validation set files.

bound F_{sup} can be derived from the insight that no annotation will be perfect given the fuzzy nature of the segmentation task. Even though closely following annotation guidelines,[5] two annotators might easily disagree on the existence or exact positions of segment boundaries. By comparing the items in the public SALAMI dataset that have been annotated twice (498 pieces in total), using one set of annotations as predictions and the other as ground truth, we calculated $F_{\text{sup},3} \approx 0.76$ for ±3 seconds tolerance, and $F_{\text{sup},0.5} \approx 0.67$ for ±0.5 seconds tolerance (note that the choice of which annotations to use as predictions and ground truth is arbitrary; the F-score does not change). Within our evaluation data subset (439 double-annotations), the results are only marginally different with $F_{\text{sup},0.5} \approx 0.68$.

8.4.4 Threshold Optimization

Peak-picking, described in Section 8.3.4, delivers the positions of potential boundaries along with their probabilities, as calculated by the CNN. The application of a threshold to those probabilities rejects part of the boundaries, affecting the precision and recall rates and consequently the F-score we use for evaluation. Figure 8.4 shows precision and recall rates as well as the F-score as a function of the threshold for the example of the `8s_std_3s` model (8 seconds of context, standard resolution, target smearing 3 seconds) at ±0.5 seconds tolerance, applied to the 100 files of the validation data set. By locating the maximum of the F-score we retrieve an estimate for the optimum threshold which is specific for each individual learned model. Since the curve for the F-score is typically flat-topped for a relatively wide range of threshold values, the choice of the actual value is not very delicate.

[5] `http://www.music.mcgill.ca/~jordan/salami/SALAMI-Annotator-Guide.pdf`, acc. May 2017



(a) 5 individual models (b) average of 5 models

Figure 8.5: Model bagging: Averaging the framewise outputs of multiple models (a) for the same file gives less noisy predictions (b) and can improve results.

8.4.5 Temporal Context Investigation

It is intuitive to assume that the CNN needs a certain amount of temporal context to reliably judge the presence of a boundary. Furthermore, the temporal resolution of the input spectra (Section 8.3.1) and the applied target smearing (Section 8.3.3) is expected to have an impact on the temporal accuracy of the predictions. In Figures 8.6 and 8.7, we compare different combinations of these model parameters, for tolerances ±0.5 seconds and ±3 seconds, respectively. For each combination, we run five experiments with different random initializations. For the case of only ±0.5 seconds of acceptable error, we conclude that target smearing must also be small: A smearing width of 1 to 1.5 seconds performs best. Low temporal spectral resolution tends to diminish results, and the context length should not be shorter than 8 seconds. For ±3 seconds tolerance, context length and target smearing are the most influential parameters, with the F-score peaking at 32 seconds context and 4 to 6 seconds smearing. Low temporal resolution is sufficient, keeping the CNN smaller and easier to train.

8.4.6 Model Bagging

As described in the previous section, for each set of parameters we trained five individual models. This allows us to improve the performance on the given data using a statistical approach: *Bagging*, in our case averaging the outputs of multiple identical networks trained from different initializations before the peak-picking stage, should help to reduce uncertainty (demonstrated in Figure 8.5). With thresholds reoptimized on the resulting boundaries (Section 8.4.4), we arrived at improvements of the F-score of up to 0.03, indicated by triangles in Figures 8.6 and 8.7.

Tables 8.1 and 8.2 show our final best results on the test set after model bagging for tolerances ±0.5 seconds and ±3 seconds, respectively. The results are set in comparison with the algorithms submitted to the MIREX campaign in 2012, 2013 and 2014, and the lower and upper bounds calculated from the annotation ground-truth (see Section 8.4.3). Especially for the lower tolerance of ±0.5 seconds, our method markedly improves over the best prior results, when using the parameter combination optimized for this scenario on the validation set.

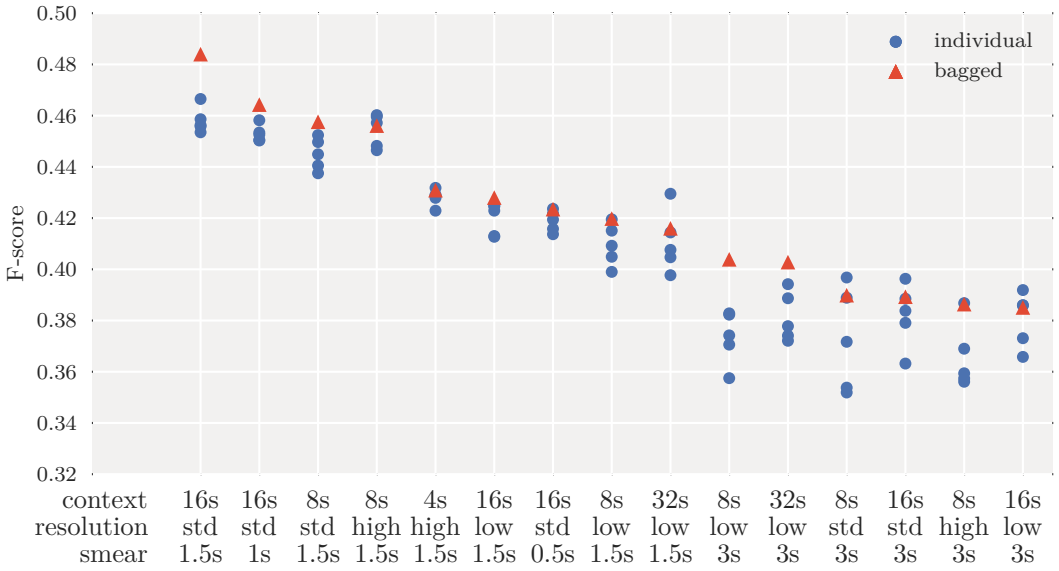


Figure 8.6: Comparison of different model parameters (context length, temporal resolution and target smearing) in terms of mean F-score on our validation set at ±0.5 seconds tolerance. Five individually trained models for each parameter combination are shown, as well as results for bagging the five models.

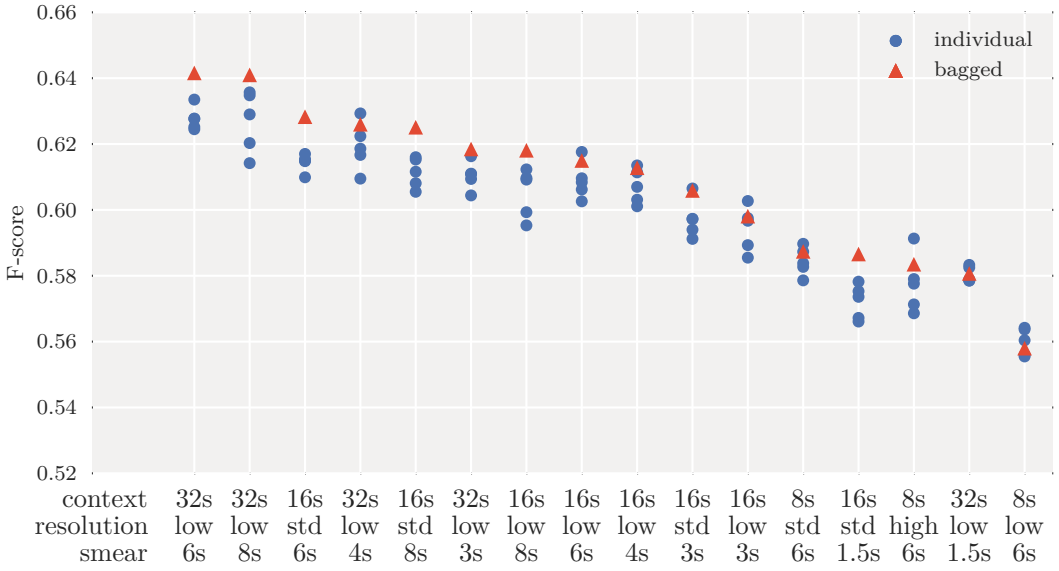


Figure 8.7: Comparison of different model parameters at ±3 seconds tolerance.

	F-score	Precision	Recall
Upper bound (est.)	0.68	–	–
16s_std_1.5s	**0.4646**	0.5553	0.4583
MP2 (2013)	0.3280	0.3001	0.4108
MP1 (2013)	0.3149	0.3043	0.3605
NB1 (2014)	0.3007	0.2542	0.4209
OYZS1 (2012)	0.2899	0.4561	0.2583
32s_low_6s	0.2884	0.3592	0.2680
KSP2 (2012)	0.2866	0.2262	0.4622
SP1 (2012)	0.2788	0.2202	0.4497
KSP3 (2012)	0.2788	0.2202	0.4497
KSP1 (2012)	0.2788	0.2201	0.4495
RBH3 (2013)	0.2683	0.2493	0.3360
NB2 (2014)	0.2647	0.2258	0.3552
NB3 (2014)	0.2641	0.2254	0.3543
RBH1 (2013)	0.2567	0.2043	0.3936
RBH2 (2013)	0.2567	0.2043	0.3936
RBH4 (2013)	0.2567	0.2043	0.3936
NJ1 (2014)	0.2274	0.1886	0.3418
CF5 (2013)	0.2128	0.1677	0.3376
CF6 (2013)	0.2101	0.2396	0.2239
SMGA1 (2012)	0.1968	0.1573	0.2943
MHRAF1 (2012)	0.1910	0.1941	0.2081
SMGA2 (2012)	0.1770	0.1425	0.2618
SBV1 (2012)	0.1546	0.1308	0.2129
Baseline (est.)	0.13	–	–

Table 8.1: Boundary recognition results on our test set at ±0.5 seconds tolerance. Our best result is emphasized and compared with MIREX campaign submissions of 2012, 2013 and 2014 evaluated on our test set.

	F-score	Precision	Recall
Upper bound (est.)	0.76	–	–
32s_low_6s	**0.6164**	0.5944	0.7059
16s_std_1.5s	0.5726	0.5648	0.6675
NB3 (2014)	0.5247	0.4530	0.6908
NB2 (2014)	0.5241	0.4520	0.6904
MP2 (2013)	0.5213	0.4793	0.6443
MP1 (2013)	0.5188	0.5040	0.5849
CF5 (2013)	0.5052	0.3990	0.7862
SMGA1 (2012)	0.4985	0.4021	0.7258
RBH1 (2013)	0.4920	0.3922	0.7482
RBH2 (2013)	0.4920	0.3922	0.7482
RBH4 (2013)	0.4920	0.3922	0.7482
SP1 (2012)	0.4891	0.3854	0.7842
KSP3 (2012)	0.4891	0.3854	0.7842
KSP1 (2012)	0.4888	0.3850	0.7838
KSP2 (2012)	0.4885	0.3846	0.7843
SMGA2 (2012)	0.4815	0.3910	0.6965
RBH3 (2013)	0.4804	0.4407	0.6076
CF6 (2013)	0.4759	0.5305	0.5102
NB1 (2014)	0.4724	0.3990	0.6605
OYZS1 (2012)	0.4401	0.6354	0.4038
NJ1 (2014)	0.4371	0.3599	0.6587
SBV1 (2012)	0.4352	0.3694	0.5929
MHRAF1 (2012)	0.4192	0.4342	0.4447
Baseline (est.)	0.33	–	–

Table 8.2: Boundary recognition results on our test set at ±3 seconds tolerance. Our best result is emphasized and compared with MIREX campaign submissions of 2012, 2013 and 2014 evaluated on our test set.

8.5 Network Examination

We are now in a similar situation as in Chapter 7: We have built a system that outperforms existing ones in the task it was trained for, but we do not know how it works. To shed some light on this, I will apply the same technique as in Section 7.5.3 (p. 131): Starting at the output for two well-chosen example files, I will partially trace back how the output was computed from the input, to understand at least one aspect of the solution. While not part of the original publication (Ullrich et al., 2014), this will refine our understanding of the difference between learned solutions and hand-designed ones.

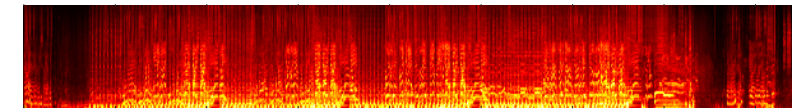
Figure 8.8a shows the spectrogram of a song in a pop concert ("The Weight" by Rachel Leber, SALAMI id 1304, shown in Figure 8.1 before), chosen for its clear instrumentation, loudness and melody changes. Figure 8.8e depicts the spectrogram of a choral piece ("Nos Autem" by Benedictine Monks of Santo Domingo De Silos, SALAMI id 228), which features a lot of short pauses, only two of them accompanied by a change of melody interpreted as a segment boundary by the human annotator.

For the pop song, the output of the network optimized for ±3 seconds tolerance (`32s_low_6s`, Figure 8.8b, shown in Figure 8.5a before) closely matches all annotated boundaries,[6] with some additional peaks such as at 3:50, when the singer speaks to the audience after the song ended. For the choral piece, the network peaks at every pause, with smaller peaks for shorter pauses (Figure 8.8f).

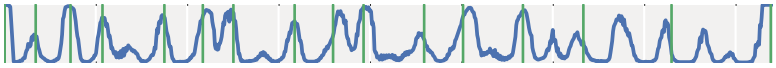
Looking at the activations of the 128 hidden units, ordered by connection strength to the output unit (Figures 8.8c and 8.8g), we see several similarities to the onset detection network of Chapter 7: (1) There are both boundary detectors (positively connected to the output, bottom part) and veto units (upper part); (2) subsets of units behave very similarly; and (3) not all units react to all of the boundaries, they specialize to catch different cues. For further investigation, I picked the unit most strongly correlated with the output for the choral piece, which peaks at each prominent pause (Figure 8.8h). On the pop song, it is selective for specific types of boundaries (Figure 8.8d): It peaks at sudden increases of loudness after a longer pause. Also note how it detects the end of the choral piece, a transition from absolute silence to our pink noise padding (Section 8.3.1), but not the end of the pop concert recording, where the upbeat of the next song is cut off by our padding.

Tracing back exactly how the hidden unit accomplishes this becomes unwieldy: Figure 8.8i shows one of the 32 feature maps of the layer before and the hidden unit's filter cross-correlated with it (the one best explaining the hidden unit activation),

[6] Compared to Figure 8.1d, I included the start of the first and end of the last segment as boundaries. We include these in training/evaluation for all segments not labelled as *silence*, consistent with MIREX: https://github.com/ismir-mirex/nemadiy/blob/ce37871/analytics/trunk/src/main/resources/org/imirsel/nema/analytics/evaluation/structure/resources/segmentRetrievalEval2.m#L22, accessed May 2017



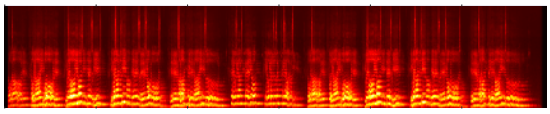
(a) input spectrogram for a pop song



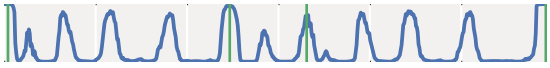
(b) network output (blue line) and ground truth (vertical green bars)

(c) penultimate layer states ordered by connection weight to output, from strongly negative (top) to strongly positive (bottom)

(d) activation of the 111$^{\text{th}}$ penultimate layer unit



(e) input spectrogram for a choral piece



(f) network output (blue line) and ground truth (vertical green bars)

(g) penultimate layer states ordered by connection weight to output

(h) activation of the 111$^{\text{th}}$ penultimate layer unit

(i) one filter of the 111$^{\text{th}}$ unit and the feature map it is applied to

Figure 8.8: Selected network weights and states for a pop song and a choral piece. The output (b, f) is a weighted combination of the penultimate layer unit activations (c, g). The one most strongly correlated with the output for the choral piece seems to detect the endings of pauses (d, h). It is a sum of cross-correlations of filters with the 32 feature maps below (i). The pop song is available for listening at `http://jan-schlueter.de/pubs/2014_ismir/`, accessed May 2017.

and this is still a few steps from the input. However, understanding how the unit detects the endings of pauses is not particularly interesting anyway – the idea is not surprising, and easy to replicate with a hand-designed algorithm (Pfeiffer, 2001).

Similar to the onset detection network, the key insight from its examination is that the network learns many slightly different variants of a handful of ideas, finely tuned to the task in a way that would be impossible to do by hand.

8.6 Extensions and Dead Ends

For an independent evaluation, we prepared a MIREX submission of our algorithm. We opted to submit two networks: the one optimized for a tolerance of ±0.5 seconds, and the one optimized for ±3 seconds, using bagging over 5 models. Compared to our previous experiments, we doubled the number of filters in the two convolutional layers to 32 and 64, respectively. On our test set, this improved F-scores from 0.4646 to 0.4755 for ±0.5 seconds, and from 0.6164 to 0.6187 for ±3 seconds.[7] The techniques described in Appendix B.3 (p. 207) enabled our submission to process files quickly enough with 5 models.[8] On all three datasets, we outperformed all other submissions in terms of boundary retrieval F-score, most notably at the shorter temporal tolerance.[9] On the SALAMI dataset, our scores were much higher than on our test subset – we assume that our training set overlaps with the SALAMI dataset used at MIREX, and consider these results to be invalid (our own test set results and the other two MIREX datasets are not affected).

Lead by Thomas Grill, we extended our approach in multiple directions. Firstly, note that with respect to the three fundamental approaches to segmentation described in Section 8.1, the CNNs in this chapter can only account for novelty and homogeneity, which can be seen as two sides of the same coin. To allow them to leverage repetition cues as well, we computed an additional input feature: a *lag matrix* that provides, for each time step, the similarities of the current spectrogram frame to a number of frames before that. Although this did not lead the network to place boundaries around long-term repetitions – probably because the training data does not strongly encourage this – it allowed the network to recognize bound-

[7] We provide the evaluation code and data to reproduce these numbers at http://jan-schlueter.de/pubs/2014_ismir/, accessed May 2017.

[8] The rules allow 24 hours for 1397 music pieces, about one minute per piece: http://www.music-ir.org/mirex/wiki/2014:Structural_Segmentation, accessed May 2017. Our submission segments a 5-minute recording in 30 s on a single core of an Intel i7 3.4 GHz CPU when using numpy/scipy, or in 5.4 s when using Theano (which includes efficient convolution algorithms).

[9] http://nema.lis.illinois.edu/nema_out/mirex2014/results/struct/mrx09/summary, http://nema.lis.illinois.edu/nema_out/mirex2014/results/struct/mrx10_1/summary, http://nema.lis.illinois.edu/nema_out/mirex2014/results/struct/sal/summary, accessed May 2017

aries from short-term regularities difficult to detect from the spectrogram alone, improving F-score at ±0.5 seconds to 0.523 (Grill and Schlüter, 2015a).

Secondly, note that many of the files are annotated by two different annotators. In this chapter, for each recording, we chose the first annotation in the dataset, discarding potentially helpful information from the second annotation. Furthermore, all files are annotated on two different hierarchical levels, focusing on different time scales. In this chapter, we used the coarse level only, but even when aiming for coarse-level predictions, the information from the fine level could be beneficial. In Grill and Schlüter (2015b), we leveraged both additional annotation sources: We included doubly-annotated files twice in the training set, once per annotation, so the network would be trained on potentially conflicting ground truth, with agreements between annotators reinforced and disagreements damped. Furthermore, we extended the network with a second output unit, training it to simultaneously predict coarse-level and fine-level boundaries. Finally, we preprocessed the spectrograms with Harmonic/Percussive Source Separation (HPSS), as I attempted for onset detection in Section 7.6 (p. 134). Evaluated on a subset of version 2.0 of the SALAMI dataset (which contains fewer trivial boundaries at file beginnings and ends), these measures improve F-score at ±0.5 seconds from 0.469 to 0.508.

In parallel to our work on improving boundary detection, we experimented with the second subtask of structural segmentation: labelling the segments to signify which ones are repetitions or variations of one another (e.g., the A, B, C labels in Figure 8.1c, p. 138). The simplest approach is to apply the boundary detector as is, compare the resulting segments, and assign them the same label if their computed similarity is above some threshold. A more advanced approach is to inform segmentation by labeling. We attempted to oversegment a piece using the CNN trained on fine-level annotations, then compare segments to merge and label them. Thomas Grill designed a segment comparison based on the cosine distance between spectrogram excerpts processed by a 2D-DCT (Grill and Schlüter, 2015c, Sec. 2.4), and any of our attempts at outperforming this by training CNNs or RNNs to perform the comparison were unsuccessful. Progress was hindered by the lack of a useful evaluation measure: The standard measures for segment labelling do not assume that two segments labelled "A" are two occurrences of similar content over time, but merely evaluate whether frames of the same ground truth label were assigned the same label by an algorithm (e.g., Levy and Sandler, 2008, Sec. V.A). Hence, for a song of four segments labelled "A", "A", "A", "B", the boundaries between the "A" segments are ignored in the evaluation. We designed a new labelling evaluation measure based on the repetition assumption, comparing ideal similarity matrices generated from ground truth and predicted segmentations. While this honours accurate segment boundaries, it ignores that same-named ground truth segments are not always repetitions. As this can only be fixed by re-annotating all data with new labelling guidelines, we eventually abandoned this line of research.

8.7 Discussion

Employing Convolutional Neural Networks trained directly on mel-scaled spectrograms, we are able to achieve boundary recognition F-scores strongly outperforming any algorithm submitted to MIREX 2012 to 2014. The networks have been trained on human-annotated data, considering different context lengths, temporal target smearing and spectrogram resolutions. As we did not need any domain knowledge for training, we expect our method to be easily adaptable to different 'foci of annotation' such as, e.g., determined by different musical genres or annotation guidelines. In fact, our method is itself an adaptation of a method for onset detection (Schlüter and Böck, 2014, as described in Chapter 7) to a different time focus, by lowering the temporal resolution of the input spectrograms and increasing the blurring of annotation targets.

In follow-up work, we have successfully explored two strategies to improve on these initial results: Providing the CNN with additional features capturing small-scale repetitions (Grill and Schlüter, 2015a), and providing additional targets in the form of secondary annotations for a subset of the music pieces, and an additional more fine-grained level of annotations for all pieces (Grill and Schlüter, 2015b).

Further improvements might be gained by following through on these strategies. Given the positive impact of adding complementary input features, it seems worthwhile to spend more effort on preparing the input data. While this is against the spirit of "end-to-end learning", music boundary detection is a high-level task that may be harder to learn from spectrograms than onset detection (Chapter 7) or singing voice detection (Chapter 9). On a related note, we leave much of the data preprocessing to the CNN, possibly using up a considerable part of its capacity. For example, the audio files in the SALAMI collection are of very different loudness, which could be fixed in a simple preprocessing step, either on the whole files, or using some dynamic gain control. Similarly, many of the SALAMI audio files start or end with noise or background sounds. A human annotator easily recognizes this as not belonging to the actual musical content, ignoring it in the annotations. The abrupt change from song-specific background noise to our pink noise padding may be mistaken for a boundary by the CNN, though. Therefore it could be helpful to apply some more intelligent padding of appropriate noise or background to provide context at the beginnings and endings of the audio.
Noting how the side task of predicting fine-level annotations also helped predicting coarse-level annotations, a promising course is to extend the network to perform further related tasks, such as downbeat detection, instrument recognition or chord and key detection – boundaries are often synchronized to the beat structure, or accompanied by a change of instrumentation or melody. Mauch et al. (2009) showed that structural segmentation can improve chord recognition, and it seems plausible that the inverse is also true.

While the examination of a trained network revealed a little bit about its inner workings – namely, that it learns multiple detectors reacting to different cues, similar to the onset detection network –, another promising direction of research is to explore the internal processing in more detail, with more recent visualization methods (Zeiler and Fergus, 2014; Simonyan et al., 2014; Springenberg et al., 2015) or by careful analysis of its output and hidden units on a wider range of examples, including failure cases. This may help to better understand the segmentation process as well as differences to existing approaches, and to refine the network architecture.

Finally, as with all methods presented in this thesis, a simple way to improve results will be to add more training data. These could be additional annotated music pieces, possibly focusing on those genres the CNN performs worse (although later work by Flexer and Grill (2016, Sec. 4.2) indicates that the most difficult genres for the CNN – denoted "classical" and "world" in the SALAMI dataset – are also least consistently annotated by humans), or even synthetic data with transitions focusing on particular musical cues under-represented in the current dataset.

9 Singing Voice Detection

In this last main chapter of the thesis, we will handle a second sequence labelling task: Detecting the presence and extents of singing voice in an audio recording. In contrast to Chapter 5, where we had plentiful finely-annotated examples to learn from, we will use the task to investigate and compare two practically important special cases: learning from a small dataset and learning from weakly-labelled data.

Specifically, towards the former, I explore the use of data augmentation to improve a CNN-based singing voice detector trained on 100 finely-annotated songs. Data augmentation enhances the diversity of a dataset with artificially perturbed examples and is especially helpful for small datasets, but has not been systematically explored for music signals before. Comparing a range of label-preserving audio transformations, I find pitch shifting to be the most helpful augmentation method, in line with recent research in speech recognition. Combined with time stretching and random frequency filtering, I achieve a reduction in classification error between 10% and 30%, slightly improving the state of the art for singing voice detection on two public datasets. As I do not employ specific task knowledge, I assume that musical data augmentation would prove helpful for other tasks as well.

Towards training on weakly-labelled data, I then develop a recipe that combines multiple-instance learning and saliency maps to train a CNN on 10,000 song-wise annotations of vocal presence, eventually achieving a temporal detection accuracy close to the new state of the art. Moreover, with saliency maps, it can even localize the spectral bins containing singing voice with precision and recall close to a recent source separation method. The developed recipe may provide a basis for other sequence labelling tasks, for improving source separation or for inspecting neural networks trained on auditory spectrograms.

Inspecting the trained networks, I find that despite their state-of-the-art performance, they rely on a fairly simple cue: diagonal or wiggly lines in a spectrogram. While not unreasonable, this leads to both missed detections and false positives. An initial attempt to overcome this with data augmentation remains unsuccessful.

This work has previously been published in two separate parts handling musical data augmentation (Schlüter and Grill, 2015) and learning from weakly-labelled data (Schlüter, 2016), respectively. Thomas Grill suggested the implementation of random frequency filtering (one of seven augmentations), but was otherwise not involved in the experiments or in writing. This chapter weaves the material of the two publications, adds more figures and explanations and three new final sections.

The remainder of this chapter is organized as follows: Section 9.1 motivates both data augmentation and learning from weakly-labelled examples. Section 9.2 describes the state of the art in singing voice detection and extraction, then reviews data augmentation and learning from weakly-labelled examples both outside of and within the music domain. Section 9.3 describes the method I used as the starting point, Section 9.4 details the augmentation methods I applied on top of it, and Section 9.5 develops a recipe for obtaining temporally accurate predictions from weakly-labelled examples. Section 9.6 presents experimental results both for musical data augmentation and for learning from weakly-labelled examples. Section 9.7 investigates how the trained network works, and Section 9.8 describes unpublished follow-up experiments. Finally, Section 9.9 highlights avenues for future research and points out alternative uses for some of the findings.

9.1 Introduction

A fundamental step in automated music understanding is to detect which instruments are present in a music audio recording, and at what time they are active. Usually, developing a system detecting and localizing a particular instrument requires a sufficiently large set of music pieces annotated at the same granularity as expected to be output by the system – no matter if the system is constructed by hand or by machine learning algorithms. However, annotating music pieces at high temporal accuracy requires skilled annotators and a lot of time. In this chapter, we will explore two directions to limit required annotation effort: reducing the number of training examples, or reducing the annotation granularity.

The problem of small training sets is that they may lack the diversity needed to constrain learning towards a solution that generalizes well. In the extreme case, it may lead to severe overfitting to the particular examples presented in training (Section 2.1.3, p. 11); in a reduced form, the solution may miss some of the inherent invariances of the task. In computer vision, a key ingredient to obtain state-of-the-art results is *data augmentation*, the technique of training and/or testing on systematically transformed examples (p. 41 in Section 2.2.5). The transformations are typically chosen to be label-preserving, such that they can be trivially used to extend the training set and encourage the system to become invariant to these transformations. This allows a classifier to learn invariances that are difficult to build into the model, and improve generalization to unseen data. As a complementary measure, at test time, aggregating predictions of a system over transformed inputs increases robustness against transformations the system has not learned to (or not been trained to) be fully invariant to.

While even earliest work on CNNs (LeCun et al., 1998a) successfully employs data augmentation, and research on speech recognition – an inspiration for many of the techniques used in Music Information Retrieval (MIR) – has picked it up as well (Jaitly and Hinton, 2013), I could only find anecdotal references to it in the MIR literature (Li and Chan, 2011; Humphrey and Bello, 2012), but no systematic treatment.

As one of the main contributions of this chapter, I devise a range of label-preserving audio transformations and compare their utility for music signals, using singing voice detection as a a benchmark task. This is a particularly good fit: The task is well-covered, but best reported accuracies on public datasets are around 90%, suggesting some leeway. Furthermore, it does not require profound musical knowledge to solve, making it an ideal candidate for training a classifier on low-level inputs. Compared to higher-level tasks, this allows observing the effect of data augmentation unaffected by engineered features, and unhindered by doubtable ground truth. For the classifier, I use CNNs, proven powerful enough to pick up invariances taught by data augmentation in other fields.

A complementary way to reduce annotation efforts is to find a way to learn from temporally coarse or even just song-wise annotations. Instrument annotations at a song level are often easily available online, as part of the tags given by users of streaming services, or descriptions or credits by the publisher. Even if not, collecting or cleaning song-wise annotations requires very little effort and low skill compared to curating annotations with sub-second granularity.

As a step towards tapping into such resources, and as the second main contribution of this chapter, I explore how to obtain high-granularity vocal detection results from low-granularity annotations. Specifically, I train a CNN on 10,000 30-second song snippets annotated as to whether they contain singing voice anywhere within, and subsequently use it to detect the presence of singing voice with sub-second granularity. To this end, I develop a recipe to improve initial results using multiple-instance learning and saliency maps. Additionally, I investigate how well the system can even pinpoint the spectrogram bins containing singing voice, instead of the time frames only. While I constrain the experiments to singing voice detection as a special case of instrument detection (and possibly the easiest), I do not assume any prior knowledge about the content to be detected, and thus expect the recipe to carry over to other instruments.

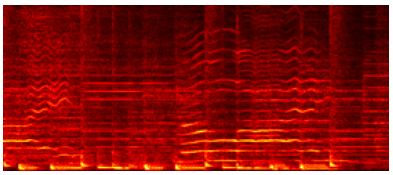
9.2 Related Work

We will now review existing work related to this chapter, both with respect to the specific tasks being addressed, and with respect to the methods employed.

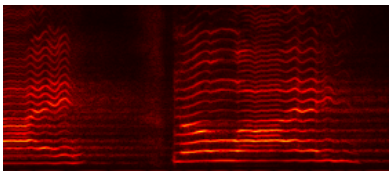
9.2.1 Singing Voice Detection

Since the initial formulation of the task by Berenzweig and Ellis (2001), many approaches for detecting the vocal parts of a music piece have been proposed in literature. As singing voice detection only serves as a test bed and solving it is not my main goal, I will limit its treatment to the state of the art, to select methods to compare results to. For a review of earlier work, see Lehner et al. (2014, Sec. 2.2).

State-of-the-art approaches for singing voice detection build on Recurrent Neural Networks (RNNs) trained on temporally accurate labels. Leglaive et al. (2015) trained a bidirectional RNN on mel spectra preprocessed with a highly tuned harmonic/percussive separation stage. They set the state of the art on the public Jamendo dataset (Ramona et al., 2008), albeit using a "shotgun approach" of training 20 variants and picking the one performing best on the test set. Lehner et al. (2015) trained an RNN on a set of five high-level features, some of which were designed specifically for the task. They achieve the second best result on Jamendo and also report results on RWC (Goto et al., 2002; Mauch et al., 2011), a second public dataset. For perspective, I will compare my results to both of these approaches.



(a) 4-second input spectrogram



(c) spectrogram of vocal signal

(b) extents of vocal parts

Figure 9.1: Singing voice detection aims to predict the temporal extents of all vocal parts (b) in a music recording (a). Singing voice extraction aims to predict a signal containing only the vocals (c) of a music recording (a).

9.2.2 Singing Voice Extraction

Singing voice detection does not entail identifying the spectrogram bins containing vocals, a side problem considered in this chapter. The closest task to this is singing voice extraction, which aims to extract a purely vocal signal from a single-channel audio recording and thus has to estimate its spectral extents (Figure 9.1). It differs from general blind source separation in that it can leverage prior knowledge about the two signals to be separated – vocals and background music.

As an improvement over Nonnegative Matrix Factorization (NMF, Smaragdis and Brown, 2003), which can only encode such prior knowledge in the form of spectral templates, a method by Rafii and Pardo (2013) called REPET uses the fact that background music is repetitive while vocals are not. Kernel-Additive Modelling by Liutkus et al. (2014) generalizes this method and uses a set of assumptions on local regularities of vocals and background music to perform singing voice extraction. A refined implementation by Liutkus et al. (2015) is publicly available; I will use it as a mark to compare my results to.

An alternative way to exploit knowledge about the signal components is to optimize a function specifically for separating vocals and background music using machine learning. A number of recent works train neural networks to predict a mask from a spectrogram: a value in $\{0, 1\}$ (binary mask) or in $[0, 1]$ (soft mask) for each spectrogram bin indicating to what proportion its magnitude reflects vocals rather than background music. Multiplying the mask with the spectrogram then recovers an estimation of the vocal signal. Huang et al. (2014) predict soft masks with RNNs, Simpson et al. (2015) predict binary masks with a 1D CNN (convolving over time only), Chandna et al. (2017) predict soft masks with a 2D CNN. The publications leave unclear which of these approaches performs best, so I use the most recent of those as an additional comparison.

9.2.3 Data Augmentation

Data augmentation generates additional data points for training or testing from existing ones by manipulating the inputs and/or targets in a way that retains their relation. Designing the manipulation methods requires domain knowledge, but methods are often generic enough to be reusable for different tasks.

For computer vision, a wealth of transformations has been tried and tested: As an early example, LeCun et al. (1998a) applied translation, scaling (proportional and disproportional) and horizontal shearing to training images of hand-written digits, improving test error from 0.95% to 0.8%. Krizhevsky et al. (2012), in an influential work on large-scale object recognition from natural images, employed translation, horizontal reflection, and colour variation. They do not provide a detailed comparison, but note that it allowed to train larger networks and the colour variations alone improve accuracy by 1 percent point. Crucially, most methods also apply specific transformations at test time (He et al., 2015).

Jaitly and Hinton (2013) pioneered the use of label-preserving audio transformations for speech recognition. They find pitch shifting of spectrograms prior to mel filtering at training and test time to reduce phone error rate from 21.6% to 20.5%, and report that scaling mel spectra either in time or frequency dimensions or constructing examples from modified Linear Predictive Coding (LPC) coefficients did not help. Concurrently, Kanda et al. (2013) showed that combining pitch shifting with time stretching and random frequency distortions reduces word errors by 10%, with pitch shifting proving most beneficial and effects of the three methods adding up almost linearly. Cui et al. (2014) combined pitch shifting with a method transforming speech to another speaker's voice in feature space and Ragni et al. (2014) combined it with unsupervised training, both targeting uncommon languages with small datasets. As far as I am aware, this comprised the full body of work on data augmentation for speech when I worked on this topic in early 2015.

In MIR, literature is even more scarce. Li and Chan (2011) observed that Mel-Frequency Cepstral Coefficients (MFCCs, p. 58) are sensitive to changes in tempo and key, and show that augmenting the training and/or test data with pitch and tempo transformations slightly improves genre recognition accuracy on the GTZAN dataset. While this is a promising first step, genre classification is a highly ambiguous task with no clear upper bound to compare results to. Humphrey and Bello (2012) applied pitch shifting to generate additional training examples for chord recognition learned by a CNN. For this task, pitch shifting is not label-preserving, but changes the label in a known way. Test accuracy slightly improves when trained with augmented data, and they observe increased robustness against transposed input. Parallel to my work (and presented at the same conference), McFee et al. (2015) proposed a software framework for musical data augmentation, but only include minimal experimental results with a limited set of augmentations.

9.2.4 Learning from Weak Labels

The idea of training on weakly-labelled data is far from new, since coarse labels are almost always easier to obtain than fine ones. In this chapter, we will investigate training a temporally accurate singing voice detector from 30-second song excerpts with excerpt-wide labels. The general framework for this setting is Multiple-Instance Learning (MIL), which we will return to in more detail in Section 9.5.1.2. Literature on this topic is vast, so I will focus on few relevant examples. For a more complete review, see Foulds and Frank (2010) and Amores (2013).

As one of the first instances, Keeler et al. (1991) trained a CNN to recognize and localize two hand-written digits in an input image of about 36×36 pixels, giving only the identities of the two digits as training targets. As a recent work closer to our setting, Hou et al. (2015) trained a CNN to detect and classify brain tumours in gigapixel resolution tissue images. As such images are too large to be processed as a whole, they propose to train on patches, still using image-level labels only. To account for the fact that not all patches in a tumour image show tumorous tissue, Hou et al. employ an expectation maximization algorithm that iteratively prunes non-discriminative patches from the training set based on the CNN's predictions.

As far as I am aware, the only prior work in MIR aiming to produce fine-grained predictions from coarse training data is that of Mandel and Ellis (2008): They trained special variants of Support Vector Machines (SVMs) on song, album or artist labels to predict tags on a granularity of 10-second clips. In contrast, I aim for sub-second granularity. Parallel to my work, Liu and Yang (2016) trained a CNN on 29-second excerpts labelled with genre and instrument tags and used it for frame-wise predictions. My work goes beyond this, developing a way to overcome a basic flaw of this approach, and to obtain predictions for each spectrogram bin.

9.3 Base Method

As a starting point for the experiments, I designed a straightforward system applying CNNs on mel spectrograms.

9.3.1 Input Features

The input signal is subsampled to 22.05 kHz, downmixed to mono and transformed to a spectrogram by a Short-Time Fourier Transform (STFT, p. 49) with Hann windows, a frame length of 1024 and hop size of 315 samples (yielding 70 frames per second). Phases are discarded, a mel filterbank with 80 triangular filters from 27.5 Hz to 8 kHz is applied (p. 51), and magnitudes are logarithmized (after clipping values below 10^{-7}, see Equation 3.10, p. 56). Finally, each mel band is normalized to zero mean and unit variance over the training set.

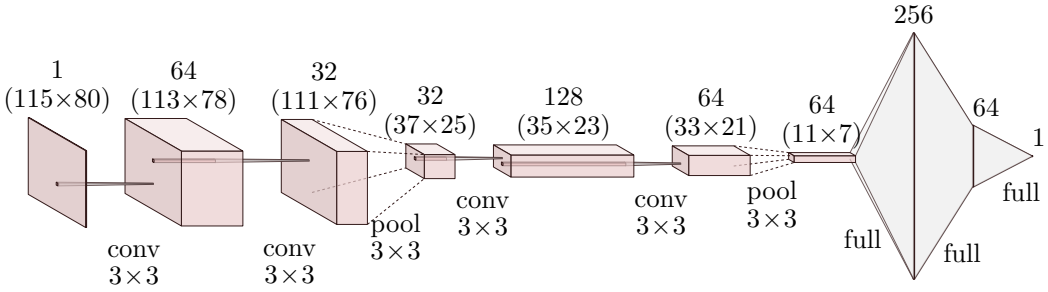


Figure 9.2: The Convolutional Neural Network architecture of the base system. Starting from a mel spectrogram excerpt, convolution and max-pooling in turns compute a set of 64 feature maps classified with a fully-connected network of 256 and 64 hidden units.

9.3.2 Network Architecture

As in Chapters 5, 7 and 8, the task is cast as a binary classification problem: Given a fixed-length mel spectrogram excerpt, predict whether there is singing voice present at the central frame. And as in Chapters 7 and 8, the model used to tackle this is a CNN of 2D convolutions, max-pooling and fully-connected layers.

Specifically, as depicted in Figure 9.2, it applies two 3×3 convolutions of 64 and 32 kernels, respectively, followed by 3×3 non-overlapping max-pooling, two more 3×3 convolutions of 128 and 64 kernels, respectively, another 3×3 pooling stage, two dense layers of 256 and 64 units, respectively, and a final dense layer of a single sigmoidal output unit. Each hidden layer is followed by the leaky rectifier nonlinearity $y(x) = \max(x/100, x)$ (see Figure 2.8c, p. 17).

The architecture is loosely copied from Simonyan and Zisserman (2015), but scaled down as the datasets in this chapter are orders of magnitude smaller. It was fixed in advance and not optimized further, as the focus of this work lies on data augmentation and learning from weakly-labelled examples, not on obtaining best possible results for singing voice detection.

9.3.3 Training Methodology

The network is trained on mel spectrogram excerpts of 115 frames (~1.6 sec) paired with a label denoting the presence of voice in the central frame. Excerpts are formed from the training songs with a hop size of 1 frame, resulting in a huge number of training examples. However, these are highly redundant: Many excerpts overlap, and excerpts from different positions in the same music piece often feature the same instruments and vocalists in the same key. Thus, instead of doing many iterations

over all possible training examples, I train the networks for a fixed number of 40,000 weight updates. While this means some excerpts are only seen once, this visits each song often enough to learn the variation present in the data. Updates are computed with stochastic gradient descent on cross-entropy error using mini-batches of 32 randomly chosen examples, Nesterov momentum of 0.95, and a learning rate of 0.01 scaled by 0.85 every 2000 updates.[1] Weights are initialized from random orthogonal matrices (see Equation 2.48, p. 35).

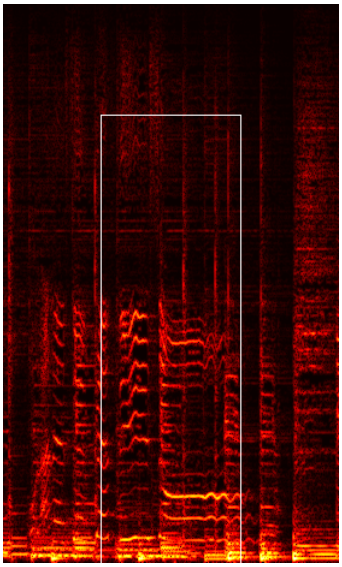
For regularization, I set the target values for the output unit to 0.02 and 0.98 instead of 0.0 and 1.0 for negative and positive examples, respectively. This avoids driving the output layer weights to larger and larger magnitudes while the network attempts to have the sigmoid output reach its asymptotes for training examples it already got correct (also noted by LeCun et al., 1998b, Sec. 4.5). I found this to be a more effective measure against overfitting than L_2 weight decay (p. 38). The same idea was proposed later by Szegedy et al. (2016b, Sec. 7) as a way to approximate regularization by label noise. Indeed, setting the targets to 0.02 and 0.98 can be seen as the expectation of randomly flipping all labels with a probability of 2%, another method to prevent overly confident network predictions (Xie et al., 2016). As a complementary measure, I apply 50% dropout to the inputs of all dense layers (p. 38 in Section 2.2.5).

All parameters were determined in initial experiments by monitoring classification accuracy at optimal threshold on validation data – i.e., during training, I regularly compute predictions on a validation set and report the accuracy obtained with the best possible classification threshold. This proved much more informative than monitoring the cross-entropy loss or the accuracy at a fixed threshold of 0.5. Still, it seemed not robust enough for early stopping (p. 37).

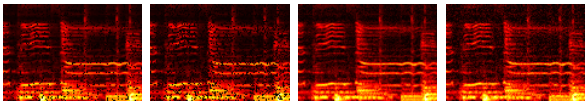
9.4 Data Augmentation

Towards the first objective of this chapter – training a singing voice detector on a comparatively small dataset – we extend the base system by augmenting the training data. Specifically, instead of training the CNN directly on mel spectrogram excerpts from the training songs, we will modify the excerpts before passing them to the network. To this end, I devised a range of augmentation methods that can be efficiently implemented to work on spectrograms: Two are data-independent, four are specific to audio data and one is specific to binary sequence labelling. All of them can be applied on-the-fly during training (some before, some after the mel-scaling stage) while collecting excerpts for the next mini-batch, and all of them have a single parameter modifying the effect strength to be varied in the experiments.

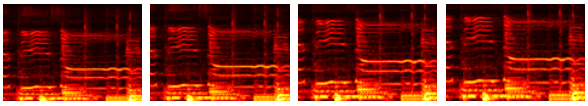
[1] The same idea of adapting hyperparameters after "mini-epochs" of 2000 mini-batches instead of after full epochs over the training set was also employed in Chapter 8.



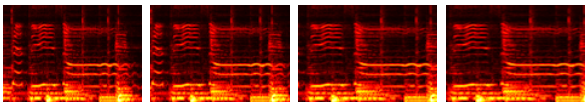
(a) Linear-frequency spectrogram excerpt. The framed part will be mel-scaled and serve as network input.



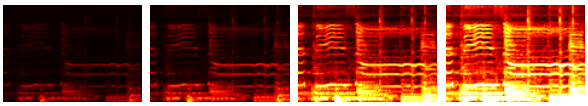
(c) Dropout (10/20%), Gaussian noise ($\sigma \in \{0.1, 0.2\}$).



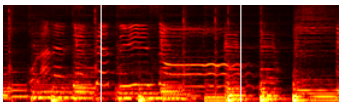
(d) Pitch shift of −20%, −10%, +10%, +20%.



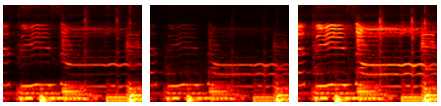
(e) Time stretch of −20%, −10%, +10%, +20%.



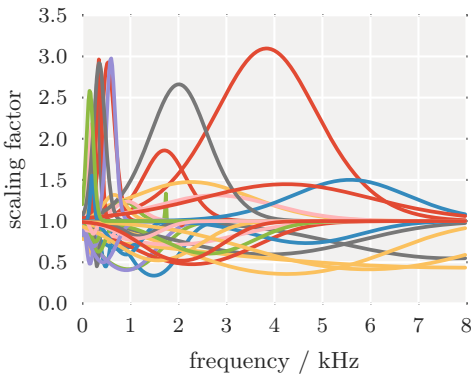
(f) Loudness of −20 dB, −10 dB, +10 dB, +20 dB.



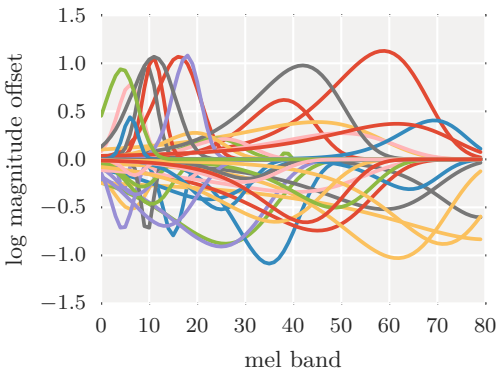
(b) Mel-scaled spectrogram.



(g) Random frequency filters.



(h) Random filter responses of up to 10 dB.



(i) Same filter responses mapped to mel scale.

Figure 9.3: Illustration of data augmentation methods on a spectrogram excerpt (0:23–0:27 of "Bucle Paranoideal" by LaBarcaDeSua).

9.4.1 Data-independent Methods

A simple way to increase a model's robustness is to corrupt training examples with random noise. I consider *dropout* – setting inputs to zero with a given probability – and additive *Gaussian noise* with a given standard deviation. This is fully independent of the kind of data we have, and I apply it directly to the standardized mel spectrograms fed into the network. Figure 9.3c shows an example spectrogram excerpt corrupted with two strengths of dropout and Gaussian noise, respectively. In contrast to the visualization, these corruptions are applied to the excerpts directly before the network, after standardization to zero mean and unit variance – thus, dropout replaces spectrogram bins by their mean over the training set, and the variance of Gaussian noise is in proportion to the variance of the training data.

9.4.2 Audio-specific Methods

Just like in speech recognition, *pitch shifting* and *time stretching* the audio data by moderate amounts does not change the label for a lot of MIR tasks. I implemented this by scaling linear-frequency spectrogram excerpts vertically (for pitch shifting) or horizontally (for time stretching), then retaining the (fixed-size) bottom central part, so the bottom is always aligned with 0 Hz, and the centre is always aligned with the label. Finally, the warped and cropped spectrogram excerpt is mel-scaled, logarithmized, standardized and fed to the network – that is, a part of the feature extraction chain (Section 9.3.1) is deferred and performed on-the-fly afterwards. Figure 9.3a shows a linear spectrogram excerpt along with the cropping borders. This would be scaled vertically and/or horizontally, holding the borders fixed. Figures 9.3d–e show the resulting mel spectrogram excerpt with different amounts of shifting or stretching. During training, the factor for each example is chosen uniformly at random[2] in a given range such as 80% to 120%, and the width of the range defines the effect strength we can vary.

A much simpler idea focuses on invariance to *loudness*: I scale linear spectrograms by a random factor in a given decibel range, or, equivalently, add a random offset to log-magnitude mel spectrograms (Figure 9.3f). Effect strength is controlled by the allowed factor (or offset) range.

As a fourth method, I apply random *frequency filters* to the linear spectrogram. Specifically, I create a Gaussian filter response $f(x) = s \cdot \exp(0.5 \cdot (x - \mu)^2/\sigma^2)$, with μ randomly chosen on a logarithmic scale from 150 Hz to 8 kHz, σ randomly chosen between 5 and 7 semitones, and s randomly chosen in a given decibel range, the width of which controls the effect strength. Figure 9.3h displays 50 such filter responses. For efficiency, responses can be mel-scaled, logarithmized (Figure 9.3i) and added to the mel spectrograms. Figure 9.3g shows three resulting excerpts.

[2] Choosing factors on a logarithmic scale might seem more natural, but did not improve results.

9.4.3 Task-specific Method

For the detection task considered here, we can easily create additional training examples with known labels by *mixing* two music excerpts together, encouraging invariance to the background. For simplicity, I only regard the case of blending a given training example A with a randomly chosen negative example B, such that the resulting mix will inherit A's label. Mixes are created from linear spectrograms as $C = (1-f)\cdot A + f\cdot B$, with f chosen uniformly at random between 0 and 0.5, prior to mel-scaling, but after audio-specific augmentations. I control the effect strength via the probability of the augmentation being applied to any given example.

9.5 Learning from Weak Labels

The second way to reduce annotation efforts considered in this chapter is to learn from song-wise annotations rather than finely annotated songs, without sacrificing temporal accuracy of the predictions. On the side, I will show how to predict not only which frames, but even which spectrogram bins contain singing voice. To this end, I change the base system (Section 9.3) to be trained on linear-frequency spectrograms, then combine ideas from multiple-instance learning and saliency maps to a three-step training procedure. In the following, I will first describe the ideas to be combined (the "ingredients"), then the training procedure (the "recipe").

9.5.1 Ingredients

The recipe combines a few methods that I will introduce up front: a singing voice detector for linear-frequency spectrograms, multiple-instance learning, and saliency maps.

9.5.1.1 Singing Voice Detection from Linear-Frequency Spectrograms

The base system described in Section 9.3 detects singing voice in mel spectrograms. To enable more accurate spectral localization of singing voice – on the granularity of frequency bins rather than mel bands – I adapt it to process linear-frequency spectrograms instead.

The feature extraction is modified to omit the mel filterbank and simply crop the spectrogram above 8 kHz instead, keeping 372 bins. Magnitudes are scaled by $\log(1+x)$ – for linear-frequency spectrograms, this gave slightly better results than $\log(\max(10^{-7}, x))$ used in the base system –, and each frequency band is normalized to zero mean and unit variance over the training set.

Applying the network architecture of the base system (Figure 9.2, p. 164) to spectrogram excerpts of 372 frequency bands instead of 80 mel bands drastically

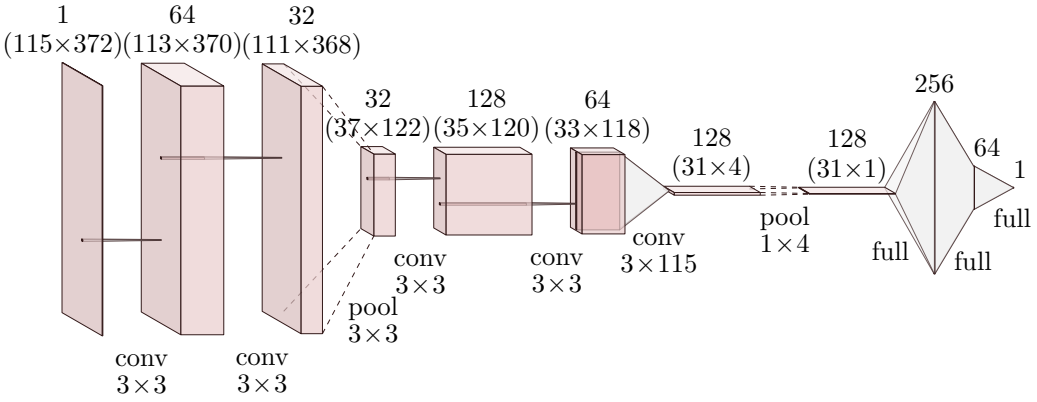


Figure 9.4: The architecture of Figure 9.2 adapted for linear-frequency input.

increases the number of parameters of the first dense layer, processing feature maps of 11×39 instead of 11×7. Preliminary experiments showed that this impairs generalization, reducing performance compared to the base system. Note that the task of the first dense layer is to integrate information over all time frames and frequency bands in its input, in contrast to a small-kernel convolution which performs a local operation only. To reduce the burden on the dense layer, we can split this task into a 1D convolutional layer integrating information over frequency bands, and the dense layer integrating information over the time frames.[3] This order reflects a stronger belief in the time-invariance of spectrogram features than in their frequency-invariance, in line with the discussion in Section 4.2.2 (p. 66).

Figure 9.4 depicts the network architecture I designed following these considerations. Like the base system, it starts with 64 and 32 3×3 convolutions, 3×3 max-pooling, 128 and 64 3×3 convolutions. At this point, the 372 frequency bands have been reduced to 118. I add 128 3×115 convolutions and 1×4 max-pooling. This way, the network learns spectrotemporal patterns spanning almost the full frequency range, applies them at four different frequency offsets and keeps the maximum activation of those four. This follows the idea of a 1D convolution over time proposed above, but additionally introduces some pitch invariance. Like the base system, it ends with three dense layers of 256, 64 and 1 unit, respectively, uses the leaky rectifier for the hidden layers and sigmoid activation for the output. As an additional deviation, each leaky rectification is preceded by batch normalization (Ioffe and Szegedy, 2015) – this slightly improved results, and was not included in the base system only because it was invented too late for Schlüter and Grill (2015).

Training follows the protocol of the base system (Section 9.3.3, p. 164).

[3] Of course we could also use further 2D convolutions and pooling, possibly even until we reach a feature map size of 1×1 and no dense layer is needed – but this would raise computational demands to a higher level than comfortably manageable with the hardware available to me.

9.5.1.2 Multiple-Instance Learning

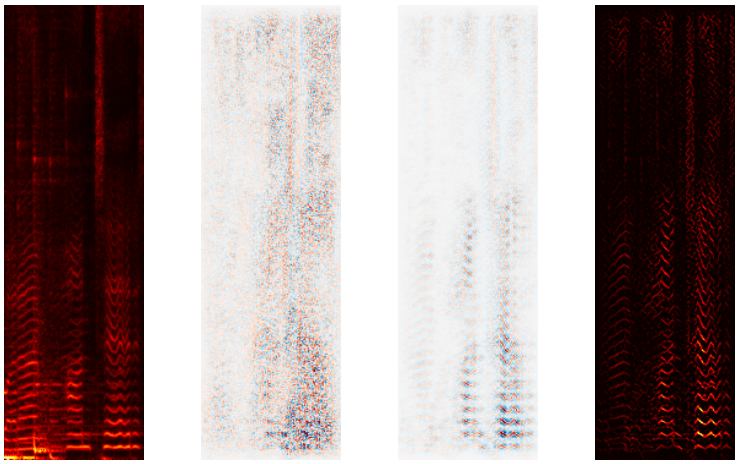
While the network is trained on short spectrogram excerpts to predict whether there is voice at the centre, we actually only have a single label per music piece. In the Multiple-Instance Learning (MIL) framework, each music piece is called a *bag*, and the excerpts we train on are called *instances*. In our setting, a bag $\mathcal{X}$ is labelled positively if and only if at least one of the instances x contained within is positive (referred to as the *standard MI assumption*, see Foulds and Frank, 2010). Formally, assuming 1 as the positive and 0 as the negative label, we have:

$$\text{label}(\mathcal{X}) = 1 \;\Leftrightarrow\; \exists_{x \in \mathcal{X}} \, \text{label}(x) = 1 \tag{9.1}$$

This gives an interesting asymmetry: If a song is labelled as "no vocals", we can infer that neither of its excerpts contains vocals. If a song contains vocals, we know that *some* excerpts contain vocals, but neither which ones nor how many.

One approach for training a neural network in this setting is based on the observation that the label of a bag is simply the maximum over its instance labels. If we define the network's prediction for a bag to be the maximum over the predictions for its instances, and the objective function to measure the discrepancy between this bag-wise prediction and the true label, minimizing it by gradient descent directly results in the following algorithm (Zhou and Zhang, 2002): Propagate all instances of a bag through the network, pick the instance that gives the largest output, and update the network weights to minimize its discrepancy with the bag label. Unfortunately, this scheme is very costly: It computes predictions for all instances of a bag, then performs an update for a single instance only. Furthermore, it is easy to overfit: It is enough for the network to produce a strongly positive output for a single chosen instance per positive bag and negative outputs for all others. Even if it does not overfit, the network is not encouraged to assign a positive output to every single positive instance, as required for our task – in contrast to Zhou and Zhang (2002), our objective is not only to correctly predict the labels of unseen bags, but even of unseen instances.

Another approach is to present all instances from negative bags as negative examples (we know they are negative) and all instances from positive bags as positive examples (they *could* be positive), and use a classifier that can underfit the training data, i.e., that may deviate from the training labels for some examples. This naive idea alone can produce good results, but it is also the basis for algorithms iteratively refining this starting point: The mi-SVM algorithm (Andrews et al., 2003) uses the predictions of the initial classifier to relabel some instances from positive bags to be negative, and alternates between retraining and relabelling until convergence. A variant proposed by Hou et al. (2015) is to prune instances from positive bags that are not clearly positive, and also iterate until convergence. For our task, the idea of improving initial results by relabelling instances will be important as well.



(a) (b) (c) (d)

Figure 9.5: Saliency mapping example: Network input (a), gradient (b), guided backpropagation (c) and its positive values (d). Best viewed on screen.

9.5.1.3 Saliency Mapping

Saliency maps for neural networks have been popularized by Zeiler and Fergus (2014) as a means of inspecting how a trained neural network forms its decisions. One of the most elegant forms computes the saliency map as the gradient of the network's output[4] with respect to its input (Simonyan et al., 2014). For a single data point, this tells how and in which direction each input feature influences the prediction for that data point. In our case, the input is a spectrogram excerpt and the gradient shows for each spectrogram bin how an infinitesimal increase would affect the probability of predicting singing voice (shown in Figure 9.5a and 9.5b, respectively). Unfortunately, for a deep neural network, an input feature can influence the output in convoluted ways: Some input may increase the output by decreasing activities in hidden layers that are negatively connected to the output.

To get a clearer picture, Springenberg et al. (2015) propose *guided backpropagation*: At each layer, only propagate the positive gradient values to the previous layer. This limits the saliency map to showing how input features affect the output by a chain of changes in the same direction. Figure 9.5c shows this for our example. A positive value (displayed in red) for a bin means increasing this bin will increase the output *by increasing activities in all layers in between*. Likewise, a negative value (depicted in blue) for a bin means that increasing it will decrease the output.

[4] Precisely, the activation of the output unit before applying the nonlinearity, as the sigmoid would dampen gradients at high activations (see Section 2.2.4.4, p. 33).

Note that the negative saliencies are not very useful: They form "halos" around the positive saliencies, indicating that the network hinges on the local contrast, and they are much less sharply localized. Assuming the hidden layers show a similar picture, this explains why ignoring negative gradients in guided backpropagation gives a sharper saliency map. To obtain a map of spectrogram bins corresponding to what the network used to detect singing voice, I keep the positive saliencies only (Figure 9.5d).

9.5.2 Recipe

With the basic ingredients in place, I will now describe the recipe to train a singing voice detector on song-wise annotations. I will first show how to use a naively trained network for temporal detection and spectral localization, and then highlight an observation about saliency maps that enables higher precision. Finally, I give a three-step training procedure that further improves results.

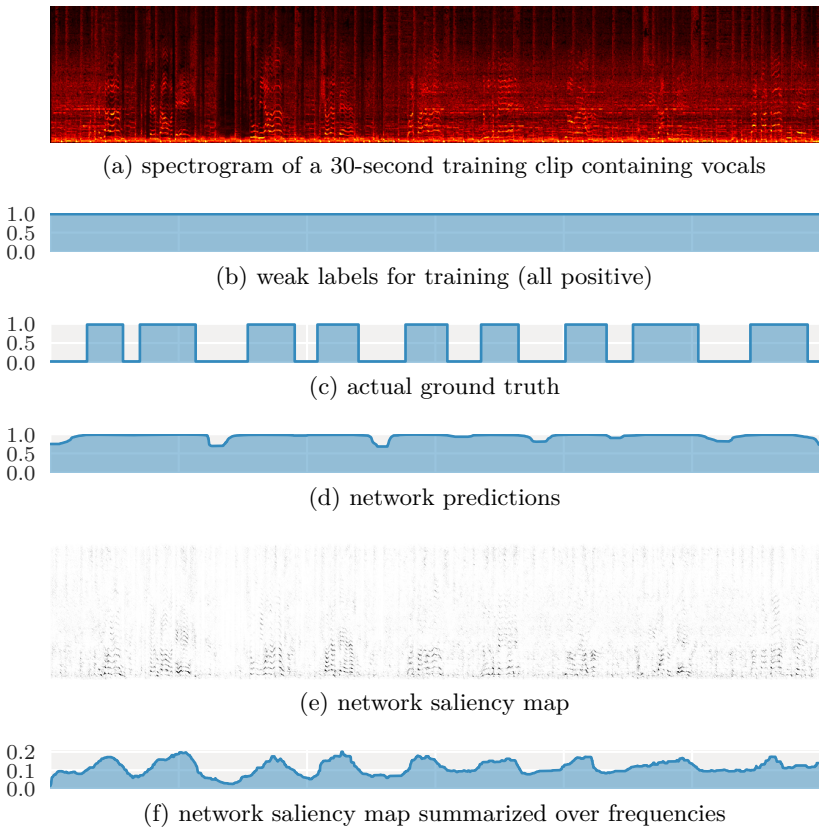
9.5.2.1 Naive Training

The easiest solution to dealing with the problem of incomplete labels – and the starting point of the recipe – is to pretend the labels were complete. I train an initial network by presenting all excerpts from instrumental songs as negative, and all excerpts from vocal songs as positive. This already works quite well: as we will see later, even on the training data, the network produces lower output for excerpts of vocal songs that do not contain voice than for those that do.

To obtain a temporal detection curve for a test song, I pass overlapping spectrogram excerpts of 115 frames through the network (with a hop size of 1 frame), recording each prediction.[5] This way, for each spectrogram frame, I obtain a probability of singing voice being present in the surrounding ±57 frames. Following Lehner et al. (2014), I post-process this curve by a sliding median filter of 56 frames (800 ms), implementing the assumption that neighbouring frames often have the same label.

For spectral localization, I also pass overlapping spectrogram excerpts through the network, each time computing the 115×372-pixels saliency map for the excerpt. To combine these into a single map for a test song, I concatenate the central frames of the saliency maps. This gives a much sharper picture than an overlap-add of the full maps. When used for vocal extraction, I apply two post-processing steps: As the saliency maps are very sparse, I apply Gaussian blurring with a standard deviation of 1 bin. And as the saliency values are very low and not proportional to the spectrogram magnitudes, I scale them to a range comparable to the spectrogram and take the elementwise minimum of the scaled map and spectrogram.

[5] See Appendix B (p. 205) for how to avoid redundant computations for overlapping excerpts.



(a) spectrogram of a 30-second training clip containing vocals

(b) weak labels for training (all positive)

(c) actual ground truth

(d) network predictions

(e) network saliency map

(f) network saliency map summarized over frequencies

Figure 9.6: Network predictions (d) overshoot vocal segments (c) because input windows only partially containing vocals were always presented as positive examples (b). Summarizing the saliency map (e) over frequencies (f) allows to correct such overshoots.

9.5.2.2 Overshoot Correction

When examining the predictions of the initial, naively trained network, I observed that it regularly overshoots the boundaries of segments of singing voice. Figure 9.6 illustrates the problem for a training example: Predictions only decline far away from vocal parts, and short pauses are glossed over. This is an artefact from naive training on weak labels: The network predicts singing voice whenever its 1.6-second input excerpt contains vocals, even if only at the edge, because such excerpts have never been presented as negative examples during training. Put differently, the network was only trained to distinguish excerpts containing vocals from such without, not to predict whether there is voice at the excerpt centre.

The saliency map provides the missing temporal information, though. By computing the saliency of the central frame of each excerpt (Figure 9.6e), we can check whether it was important for that excerpt's prediction. Summing up the saliency map over frequencies (Figure 9.6f) gives an alternative prediction curve that can be used to improve the precision of temporal detection. In the next section, I will use this observation to improve the network.

9.5.2.3 Self-Improvement

A basic idea described in Section 9.5.1.2 (p. 170) is that the naively trained network could be used to relabel the positive training instances, which necessarily contain false positives (compare Figure 9.6b to 9.6c). The intuition is that correcting even just a few of those and retraining the network should improve results.

I tried several variants of this idea: Relabelling by thresholding the initial network's predictions (using a low threshold to only relabel very certainly false positives), weighting positive examples by the initialized predictions (so confidently positive examples would affect training more than others), or removing positive examples the initial network is not confident about. However, the only effect was that the bias of the retrained network was lower, and iterating such a scheme converged to a network predicting "no" all the time. In hindsight, this is not surprising: The only positive instances we relabel this way are those the network got correct with naive training already, so it will not learn anything new when retraining.

I found a single scheme that does not deteriorate results: Training a second network to output the temporally smoothed predictions of the initial network for positive instances, and the actual labels for negative instances. It does not result in better predictions either, but in much clearer saliency maps with less noise in non-vocal sections. Using the technique of Section 9.5.2.2, I find that for such a second network, the summarized saliency maps alone actually provide a better temporal detection curve than the network output, as it does not suffer from overshoot.

Iterating the latter scheme by retraining on the second network's predictions does not help. However, we can train a third network to output the temporally smoothed summarized saliency maps of the second network for positive instances, again keeping negative instances at their true labels. To bring them to a suitable range for training (i.e., between 0 and 1), I scale them by 10 and apply the tanh function. Finally, this third network gives predictions that do not need any overshoot correction. It can be seen as finding a more efficient way of computing the summarized saliency map of the second network.

Put together, the recipe consists of: (1) Training a first network (subsequently called CNN-α) on the weak instance labels, (2) training a second network (CNN-β) on the predictions of CNN-α, (3) training a third network (CNN-γ) on the summarized saliency maps of CNN-β.

	total size	training size	ground truth
In-House A^+	9751 clips, 81 h	all 9751 clips	clip-wise annotations
In-House A	188 clips, 1.6 h	100 clips, 0.8 h	subsecond-wise annotations
In-House B	135 songs, 9.3 h	65 songs, 4.3 h	subsecond-wise annotations
Jamendo	93 songs, 6.2 h	61 songs, 4.0 h	subsecond-wise annotations
RWC	100 songs, 6.8 h	80 songs, 5.4 h	subsecond-wise annotations
ccMixter	50 songs, 3.2 h	not trained on	separate vocal tracks
MedleyDB	52 songs, 2.9 h	not trained on	separate vocal tracks

Table 9.1: Overview over the singing voice datasets used.

9.6 Experimental Results

We will now empirically investigate how well the data augmentation methods proposed in Section 9.4 help training a singing voice detector on comparatively small datasets, and how well the recipe of Section 9.5 allows training a singing voice detector on weakly-labelled data, in this order. Before, I will introduce the datasets and evaluation measures used, which overlap between the two series of experiments.

9.6.1 Datasets

Table 9.1 gives an overview over the datasets employed, described in detail below. First of all, I curated a dataset for training a network on weakly-labelled data:

In-House A^+ is based on 10,000 30-second preview snippets from an online music store, covering a very wide range of genres and origins, from 10,000 different artists (one song per album of the training set of Chapter 6). Using a custom web interface, I had 5 annotators sort 2,000 snippets each into vocal and non-vocal ones, where "vocal" was defined to be "any use of human vocal chords". Annotators were allowed to skip examples they were very unsure about or that only contained voice-like sound effects, leaving 9751 annotated clips, 6772 of which were labelled to contain vocals.
The annotation interface was designed to allow fast annotation: It plays each new clip automatically, allows to label a clip as "vocal" before reaching its end, to advance playback or go back in steps of 1 second, and control everything by keyboard. On average, annotators took 8.7 s to decide on a 30-second clip (5.1 s per clip for the fastest, 12.5 s per clip for the slowest annotator).
To check inter-annotator agreement, I had 100 clips from the dataset be labelled by 6 annotators. Of those, annotators agreed on 97 clips (9 of those were skipped by some annotators, but agreed on by the others). The remaining 3 were skipped by at least one annotator and disagreed on by others.

Four datasets are used for training or evaluating fine-grained temporal detection:

In-House A is a subset of *In-House A*$^+$ formed as follows. The $97-9=88$ clips all 6 annotators agreed on and did not skip serve as a well-proven validation set, 100 more randomly chosen clips serve as a training set (there is no test set; this dataset will be used for development only). I annotated all 188 clips with sub-second granularity. For comparison to the statistics above, on average, this fine-grained annotation took 125 s per clip (between 33 s and 522 s).

In-House B consists of 139 full-length rock songs. While being far less diverse, this dataset features a lot of electric guitars that share characteristics with singing voice. I use 65 files for training, 74 for validation (again, no test set).

Jamendo has 93 full-length Creative Commons songs collected and annotated by Ramona et al. (2008). For comparison to existing results, I follow the official split of 61 files for training and only 16 files each for validation and testing.

RWC: The RWC-Pop collection by Goto et al. (2002) contains 100 pop songs, with singing voice annotations by Mauch et al. (2011). To compare results to Lehner et al. (2015), I use the same 5-fold cross-validation split (personal communication). In contrast to the other datasets, RWC contains duplicate artists, and the split unfortunately does not respect this – thus, results for this dataset might be slightly optimistic.

Finally, two datasets with separated vocals serve for evaluating spectral localization:

ccMixter, collected by Liutkus et al. (2014), contains 50 songs from `ccmixter.org`.

MedleyDB, compiled by Bittner et al. (2014), consists of 122 songs. I only use the 52 songs that feature vocals (singer or rapper) and do not have bleed between tracks. Downmixes are provided, I obtain the corresponding vocal tracks by mixing all vocal stems per song.

9.6.2 Evaluation

For temporal detection of singing voice, at test time, each network outputs a value between 0 and 1 indicating the probability of voice being present at the centre of a 1.6-second spectrogram excerpt. Feeding maximally overlapping excerpts, we obtain a sequence of 70 predictions per second. Following Lehner et al. (2014), I apply a sliding median filter of 800 ms to smooth the output.

For the data augmentation experiments, I apply a threshold to the prediction curve to obtain binary predictions. I compare these predictions to the ground truth labels to obtain the number of true and false positives and negatives, accumulated over all test songs. While several authors use the F-Score to summarize results, I follow the argument of Mauch et al. (2011) that a task with over 50% positive

examples is not well-suited for a document retrieval evaluation measure. Instead, I report the classification error, recall and specificity (recall of the negative class).

When training on weakly-labelled examples, there is no finely-annotated validation data we can use to choose the classification threshold. For this setting, I therefore opt to assess the quality of the detection curves, rather than hard classifications. Specifically, I compute two measures: The Area Under the Receiver Operating Characteristic curve (AUROC, short AUC), and the classification accuracy at the optimal threshold for the test set. The former gives the probability that a randomly drawn positive example gets a higher prediction than a randomly drawn negative example, and the latter gives a lower bound on the error.

Finally, for spectral localization, there is no established evaluation measure. The closest to this is singing voice extraction, which is evaluated with standard source separation measures (e.g., the source to interference ratio, see Vincent et al., 2006). However, developing a fully fledged source separator is out of scope for this work. I therefore resort to comparing the saliency map produced by the network for a mixed signal to the spectrogram of the known pure-vocal signal. Specifically, I compute a generalization of precision, recall and F-score towards real-valued (instead of binary) targets: For a predicted saliency map P_{ij} and pure-vocal spectrogram T_{ij}, I define the total amount of true positives as $t = \sum_{i,j} \min(P_{ij}, T_{ij})$. I then obtain precision as $p = t/\sum_{i,j} P_{ij}$, recall as $r = t/\sum_{i,j} T_{ij}$, and F-score as $f = 2pr/(p+r)$.

9.6.3 Influence of Data Augmentation

In the first series of experiments, I evaluate if and by how much data augmentation helps training a singing voice detector from between 50 minutes and 5.4 hours of training data, much less than the 15 hours we had available for music and speech detection in Chapter 5 (p. 80), or the 12,000 hours used for modern speech recognition systems (Amodei et al., 2015, Tab. 9; of course a much harder problem).

I first compare the proposed augmentation methods in isolation at different augmentation strengths on the internal datasets *In-House A* and *In-House B*, to determine how helpful they are and how to set their parameters, and then combine the best methods. Secondly, I assess the use of augmentation at test time, both for networks trained without and with data augmentation. Finally, I evaluate the best setting on the two public datasets *Jamendo* and *RWC*, comparing against the base system and the state of the art.

Train-time augmentation: To compare the augmentation methods, I train the base system with each of the seven different augmentations (Section 9.4, p. 165) on each of the two internal datasets, and evaluate it on the (non-augmented) validation files. I report classification errors at the optimal binarization threshold to enable a direct comparison of augmentation methods unaffected by threshold estimation.

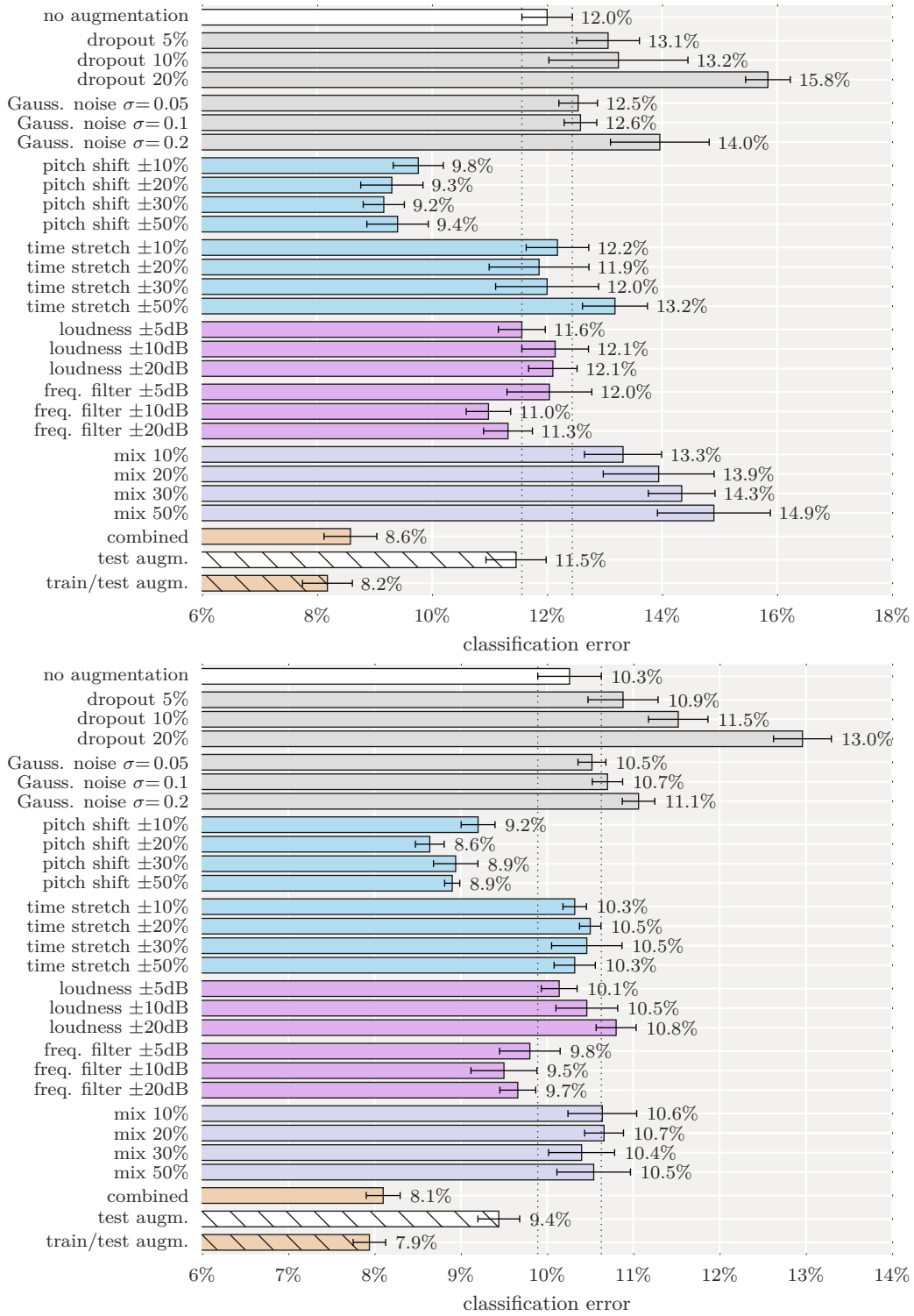


Figure 9.7: Classification error for different augmentation methods on internal datasets (top: *In-House A*, bottom: *In-House B*). Bars and whiskers indicate the mean and its 95% confidence interval computed from five repetitions of each experiment.

Figure 9.7 depicts the results. The first row in each of the two panels gives the result of the base system without any data augmentation. All other rows except for the last three per panel show results with a single data augmentation method at a particular strength. Dotted vertical lines indicate the 95% confidence interval of the base system performance computed from five repetitions with random network initializations, to facilitate comparison of results to the base system.

Corrupting the inputs even with small amounts of noise clearly just diminishes accuracy. Possibly, its regularizing effects (An, 1996) only apply to simpler models, as it is not used in recent object recognition systems either (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; He et al., 2015); or it is useful only when evaluating on noisy data (cf. Amodei et al., 2015, Sec. 5.2). Pitch shifting in a range of ±20% or ±30% gives a significant reduction in classification error of up to 25% relative. It seems to appropriately fill in some gaps in vocal range uncovered by the small training sets. Time stretching does not have a strong effect, indicating that the cues the network picked up are not sensitive to tempo. Similarly, random loudness change does not affect performance. Random frequency filters give a modest improvement, with the best setting at a maximum strength of 10 dB. Mixing in negative examples clearly hurts, but a lot less severely on the second dataset. Presumably this is because the second dataset is a lot more homogeneous, and two rock songs mixed together still form a somewhat realistic example, while excerpts randomly mixed from the first dataset are far from anything in the test set. I hoped mixing examples would drive the network to recognize voice irrespectively of the background, but apparently this is too hard or besides the task.

The third from last row per panel shows performance for combining pitch shifting of ±30%, time stretching of ±30% and frequency filtering of ±10 dB. While error reductions do not add up linearly as in speech recognition experiments of Kanda et al. (2013), it yields a ~6% relative improvement over pitch shifting alone.

Test-time augmentation: In object recognition systems, it is customary to also apply a set of augmentations at test time and aggregate predictions over the different variants (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; He et al., 2015). Here, I average network predictions (before temporal smoothing and thresholding) over the original input and pitch-shifted input of −20%, −10%, +10% and +20%. Unsurprisingly, other augmentations were not helpful at test time: Tempo and loudness changes hardly affected training either, and all remaining methods corrupt data.

The last two rows per panel in Figure 9.7 show results with this measure when training without data augmentation and our chosen combination, respectively. Test-time augmentation is beneficial independently of train-time augmentation, but increases computational costs of doing predictions.

	Jamendo			RWC		
	Error	Recall	Spec.	Error	Recall	Spec.
Lehner et al. (2015)	10.6%	90.6%	–	7.7%	93.4%	–
Leglaive et al. (2015)	8.5%	92.6%	–	–	–	–
Ours w/o augmentation	9.4%	90.8%	90.5%	8.2%	92.4%	90.8%
train augmentation	8.0%	91.4%	92.5%	7.4%	93.6%	91.0%
test augmentation	9.0%	92.0%	90.1%	8.2%	93.4%	89.4%
train/test augmentation	7.7%	90.3%	94.1%	7.3%	93.5%	91.6%

Table 9.2: Results with data augmentation on Jamendo and RWC.

Comparison to state of the art: While the main objective was to evaluate if and which data augmentation methods are helpful for music data, to set results in perspective, I train and evaluate the base system on two public datasets, adding the combined train-time augmentation, test-time pitch-shifting, or both. For *Jamendo*, I train on the training set, optimize the classification threshold on the validation set and evaluate on the test set. For *RWC*, I employ 5-fold cross-validation (with the split of Lehner et al., 2015), using four folds for training and one for testing, and evaluate using the optimal threshold determined for *In-House A*.

As can be seen in Table 9.2, on both datasets results slightly improve upon the state of the art. This shows that augmentation did not only help because the proposed base system was a weak starting point, but actually managed to raise the bar. Furthermore, in line with the results on the two internal datasets, train-time augmentation helps more than test-time augmentation, and combining both measures further improves results. Note that the methods I compared to would probably also benefit from data augmentation, possibly surpassing my results.

9.6.4 Temporal Detection from Weak Labels

In the second series of experiments, I evaluate if and how well we can train a singing voice detector on clip-wise annotations, an alternative to training on finely-annotated, but small datasets for the common goal of limiting annotation efforts. Specifically, I will follow the self-improvement recipe for weakly-annotated data (Section 9.5.2.3, p. 174) on the large *In-House* A^+ dataset and compare results to a baseline network trained on *In-House A*, a subset of finely-annotated clips.

In contrast to the previous experiments, networks will be trained on linear-frequency spectrogram excerpts (Section 9.5.1.1, p. 168) to prepare for the side task of predicting the time-frequency bins containing singing voice (in the next section). As a further change, I assume that there is no finely-annotated validation set we can use to optimize any classification thresholds, so we will change the evaluation

	In-House A		Jamendo		RWC	
	AUC	acc.	AUC	acc.	AUC	acc.
CNN-α predictions	.911	.888	.913	.865	.879	.856
summed saliencies	.955	.896	.930	.849	.912	.843
CNN-β predictions	.922	.888	.923	.875	.890	.861
summed saliencies	.970	.916	.955	.894	.936	.883
CNN-γ predictions	.970	.915	.960	.901	.939	.887
summed saliencies	.965	.914	.950	.898	.931	.884
CNN-fine predictions	.979	.930	.951	.880	.947	.882
summed saliencies	.969	.909	.948	.885	.937	.883

Table 9.3: Temporal detection results for the three steps of self-improvement on weak labels (Section 9.5.2.3) as well as a network trained on fine labels. Networks are trained on In-House A^+/A, and for Jamendo/RWC, the full datasets are used for testing, so results are not comparable to Table 9.2.

measures as described in Section 9.6.2 (p. 176). Thus, we cannot reuse results from the previous section as a baseline. To establish a new baseline, I train a network on the 100 finely-annotated training clips of *In-House A*, denoted as CNN-fine.

This baseline is set in comparison to CNN-α, CNN-β and CNN-γ corresponding to the three steps of the self-improvement recipe, trained on *In-House* A^+. Recall that *In-House A* is a finely-annotated subset of *In-House* A^+, with 100 training clips and 88 validation clips. To use the 88 validation clips as a common evaluation set, I exclude them from the training data for CNN-α to CNN-γ, leaving 9663 weakly-annotated training clips.

Table 9.3 shows the results. For each network, I assess the quality of its predictions and of its summarized saliency map, which should correct possibly overshooting predictions (Section 9.5.2.2, p. 173). Networks are evaluated on the validation set of *In-House A* as well as the full *Jamendo* and *RWC* datasets. We can see that for CNN-α, summarized saliencies are better than its direct predictions in terms of AUROC, but tend to be worse in terms of optimal classification error. CNN-β, which is trained on the predictions of CNN-α, performs strictly better and gains a lot when using summarized saliencies instead of predictions. CNN-γ is trained to predict the saliencies of CNN-β and matches or even outperforms those. Its saliency maps do not provide a benefit. CNN-fine performs comparably to CNN-γ: It is strictly better on *In-House A*, but not on the public test sets, indicating that CNN-γ profits from its larger training set despite the weak labels, achieving better generalization. The summarized saliencies of CNN-fine do not provide any improvement, as to be expected: There is no overshoot they could correct.

	ccMixter			MedleyDB		
	prec.	rec.	F_1	prec.	rec.	F_1
baseline	.324	.947	.473	.247	.955	.361
Liutkus et al. (2015)	.597	.681	.627	.416	.739	.484
Chandna et al. (2017)	.568	.599	.566	.438	.732	.505
CNN-α	.575	.651	.603	.467	.637	.497
CNN-β	.565	.763	.643	.522	.618	.529
CNN-fine	.552	.795	.646	.494	.669	.528

Table 9.4: Spectral localization results for the baseline of just predicting the spectrogram of the mix, two voice/music separation methods, and saliency maps of three networks.

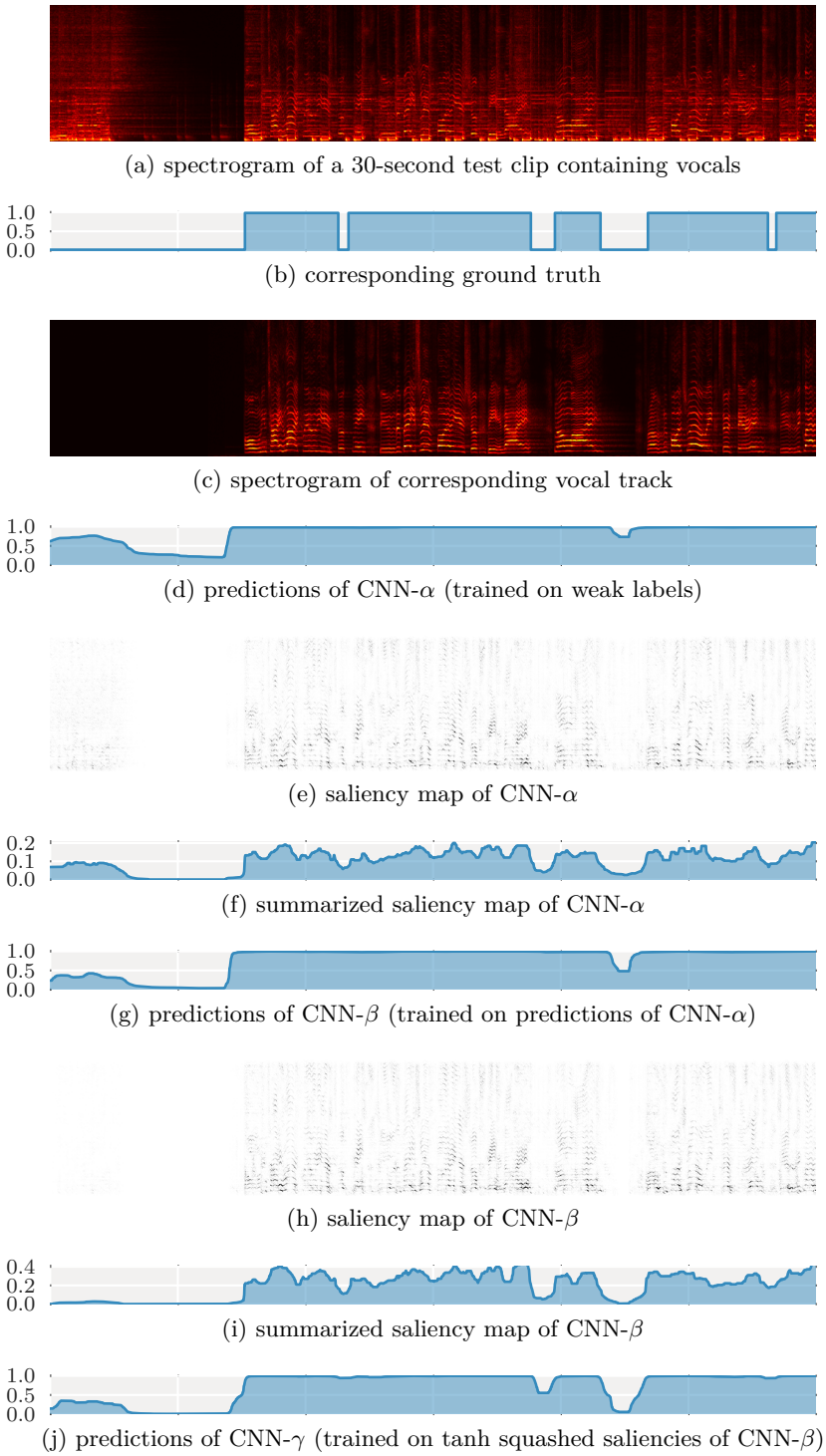
Figure 9.8 provides a qualitative comparison of the predictions for the three networks. In line with the quantitative results, we see that CNN-β is closer to the ground truth than CNN-α (especially in the first part), that the summarized saliency maps fix the temporal overshoot, and that CNN-γ approximates the summarized saliencies of CNN-β (scaled and squashed as described on p. 174).

Note that for all networks, training includes the best-performing combination of train-time augmentation found in the previous section (pitch shifting and time stretching of ±30%, frequency filtering of ±10 dB). Without data augmentation, CNN-fine is not competitive with the much larger weakly-annotated training set. To have similar conditions between training on finely- and weakly-annotated data, I use data augmentation for CNN-α to CNN-γ as well.

9.6.5 Spectral Localization from Weak Labels

Finally, I investigate how well networks trained for temporal voice detection can predict the spectrogram bins containing voice, by comparing their saliency maps to the spectrogram of the vocal tracks available for *ccMixter* and *MedleyDB*. I evaluate the saliency maps of CNN-α, CNN-β and CNN-fine, post-processed as described in Section 9.5.2.1 (p. 172), computing the precision, recall and F-score as described in Section 9.6.2 (p. 176). CNN-γ is omitted as it is tailored for temporal detection and will not produce better saliencies than CNN-β.

Results are shown in Table 9.4. As in temporal detection, CNN-β has an edge over CNN-α, and is close to CNN-fine. As a simple baseline, I use the spectrogram of the mix, which gives a low precision (as it includes a lot of non-vocal sound), but 100% recall (as it includes all vocals) – in practice, recall only reaches 95% since the songs are mastered and the mix is sometimes lower than the vocal track. The



(a) spectrogram of a 30-second test clip containing vocals

(b) corresponding ground truth

(c) spectrogram of corresponding vocal track

(d) predictions of CNN-α (trained on weak labels)

(e) saliency map of CNN-α

(f) summarized saliency map of CNN-α

(g) predictions of CNN-β (trained on predictions of CNN-α)

(h) saliency map of CNN-β

(i) summarized saliency map of CNN-β

(j) predictions of CNN-γ (trained on tanh squashed saliencies of CNN-β)

Figure 9.8: Qualitative demonstration of the self-improvement recipe (p. 174) for a single test clip (0:24 to 0:54 of "Vermont" by "The Districts", part of the MedleyDB dataset). For an interactive version, see `http://jan-schlueter.de/pubs/2016_ismir/thedistricts`, acc. June 2017.

CNNs achieve better F-scores with lower recall and much higher precision than the baseline, indicating that they correctly ignore most of the instrumental background. For additional points of comparison, I extract vocal signals with KAML (Liutkus et al., 2015) and the RNN of Chandna et al. (2017) and compute their spectrograms, clipped to 8 kHz to be comparable to the network's saliency maps. They also achieve much higher precision than the baseline, but fall slightly behind CNN-β and CNN-fine in terms of F-score.

While these results look promising, it should be noted that the saliency maps will not necessarily be better for source separation: A lot of the improvement in F-score hinges on the fact that the saliency maps are often perfectly silent in passages that do not contain vocals, while KAML still extracts parts of background instruments. This gives an advantage in precision. For passages that do contain vocals, the post-processed saliency maps include more instrumental interference than KAML.

9.7 Network Examination

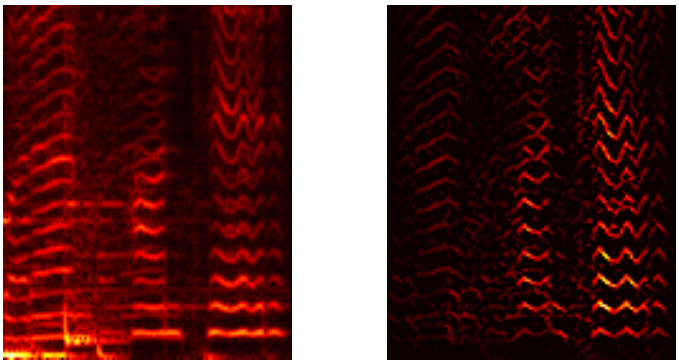
As in the previous chapters, we will inspect the trained networks to understand how they solve their task. Instead of visualizing filters or activities in hidden layers, I will use the saliency maps and a failure case analysis to form a hypothesis on what the network does, and verify this experimentally.[6]

Figure 9.9 shows an enlarged excerpt of the spectrogram input and corresponding saliency map already displayed in Figure 9.5 (p. 171), for a network trained on weakly-labelled linear-frequency spectrogram excerpts. By definition, scaling the saliency map by an infinitesimal factor and adding it to the spectrogram will increase the network's output compared to the original input; subtracting it will decrease the output.[7] We see that the output strongly depends on the presence of vibrato: Even for the fundamentals and the first harmonic (the bottommost lines), which are comparatively flat in the input spectrogram, adding or enhancing periodic frequency fluctuations would increase the network's confidence in predicting singing voice. In the leftmost part, the network is more sensitive to the higher harmonics than to the fundamental, possibly because the former have a larger slope.

The saliency maps show that voice with vibrato is detected by its vibrato. For further insights, we investigate what happens for voice without vibrato. Figure 9.10a shows an excerpt of a song with a note held by the singer for some seconds with slowly modulated pitch. Both the network output and saliency map (Figures 9.10d–e) miss this note. Notably, KAML misses the note as well (Figure 9.10f).

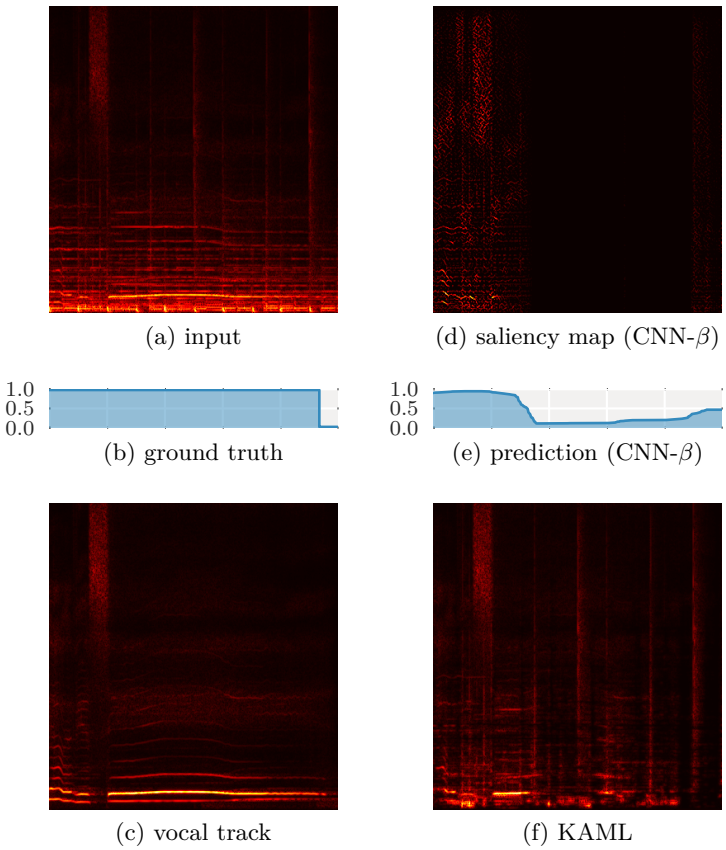
[6] Saliency maps by guided backpropagation (Springenberg et al., 2015) were not used in previous chapters because they had not been invented yet; see the timeline on p. 43.

[7] Since the saliency map is based on the gradient of the prediction wrt. the input, in general, this behaviour is only guaranteed for infinitesimal changes. As a network with (leaky) linear rectifiers is piecewise linear, here it will hold at least until the next breakpoint.



(a) input (b) saliency map (CNN-α)

Figure 9.9: Network input and corresponding saliency map (positive values of guided backpropagation, see p. 171), shown up to 3 kHz.

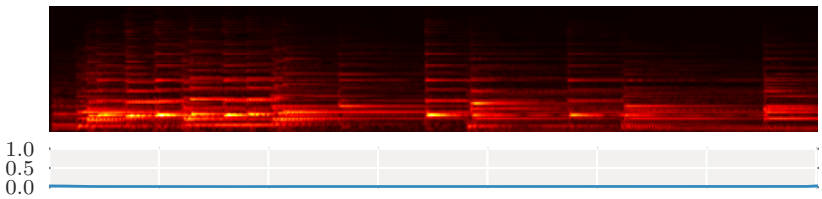


(a) input (d) saliency map (CNN-β)

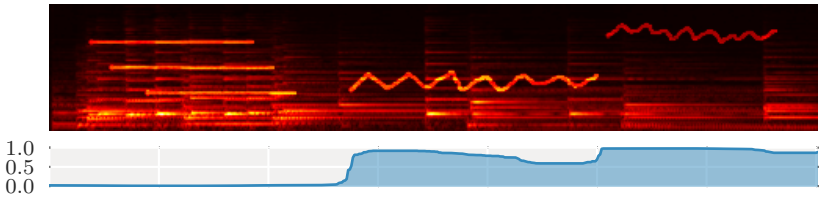
(b) ground truth (e) prediction (CNN-β)

(c) vocal track (f) KAML

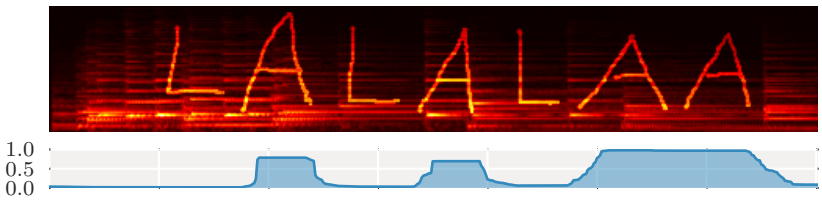
Figure 9.10: Both the CNN (d, e) and KAML (f) miss a long drawn note sung without vibrato (a–c) in a test excerpt (1:31 to 1:36 of "Air Traffic" by "Clara Berry and Wooldog", part of MedleyDB). For an interactive version, see http://jan-schlueter.de/pubs/2016_ismir/claraberryandwooldog, accessed June 2017.



(a) mel spectrogram and predictions for a piano recording



(b) artificially modified mel spectrogram and predictions



(c) artificially modified mel spectrogram and predictions

Figure 9.11: Artificial sloped lines added to the spectrogram of a 7-second piano recording are mistaken for singing voice by a trained CNN. For an interactive version, see http://jan-schlueter.de/pubs/phd/horse or http://github.com/f0k/singing_horse, accessed July 2017.

A possible hypothesis is that the network only learned to detect oscillating lines, or even only sloped lines (noting that vibrato is just a sequence of upward and downward sloped line segments). To verify this hypothesis, I perform a drastic experiment: Starting with a spectrogram that is purely instrumental, I artificially modify the magnitudes with an image editor to resemble differently sloped lines and compute predictions with the network. Figure 9.11 shows the result for a short piano excerpt and two modified versions, this time using a network from the first series of experiments that has been trained on mel spectrogram excerpts of the *Jamendo* dataset. While the network does not detect vocals in the original spectrogram (Figure 9.11a) or for artificial horizontal lines (Figure 9.11b, left part), both artificial oscillating lines (Figure 9.11b, right part) and upward or downward slopes (Figure 9.11c) are mistaken for voice, despite not sounding like voice at all when resynthesized (try the interactive version of the figure).

Implications of these findings will be discussed in Section 9.9.

9.8 Extensions and Dead Ends

Next to the experiments published in Schlüter and Grill (2015) and Schlüter (2016) and presented in this chapter, I used singing voice detection as an example task to try several ideas ultimately not resulting in a publication, but still worth discussing. For reproducibility, most experiments use the public *Jamendo* dataset, and code and instructions are made available at `http://github.com/f0k/ismir2015` in the `phd_extra` branch, and at `http://jan-schlueter.de/pubs/phd/singing.zip`.[8]

Architecture comparisons: In the experiment series on data augmentation, a network based on 2D convolutions of mel spectrogram excerpts achieved 8.6% classification error on the *In-House A* dataset with the best combination of train-time augmentation and no test-time augmentation (Figure 9.7, p. 178). In the experiments for training on weakly-labelled examples, the baseline network achieved 7.0% classification error on the same dataset under the same conditions (accuracy .930, Table 9.3, p. 181). It differs in three important aspects: (1) it is trained on linear-frequency spectrogram excerpts rather than mel spectrogram excerpts; (2) it uses batch normalization; (3) its architecture features a convolution covering almost all frequency bands in front of the dense layer (explained in detail in Section 9.5.1.1, p. 168).
To understand what causes the leap in performance, I applied different combinations of these modifications to a network trained on the *Jamendo* dataset. The original architecture yields an error of 8.0% (Table 9.2, p. 180). Batch normalization does not change results. Switching to the more complex architecture (Figure 9.4, p. 169), but using mel spectrograms, results in 7.5(±0.2)% without, and 7.3(±0.5)% with batch normalization.[9] Finally, training it on linear-frequency spectrograms yields 7.0(±0.3)% without, and 7.4(±0.3)% with batch normalization. It seems the improvement partly stems from changing the architecture, not only from using linear-frequency spectrograms.

Learned magnitude transformation: A subtlety between the experiment series on mel spectrograms and the one on linear-frequency spectrograms is that I transform magnitudes with $\log(\max(10^{-7}, x))$ for the former and $\log(1+x)$ for the latter. I arrived at these settings in preliminary experiments on *In-House A*, but such hand-tuning is cumbersome and might not lead to the best solution. As an alternative, I tried optimizing the magnitude transformation as part of the network: Before the first convolution, I put an elementwise $\log(1+10^a \cdot x)$ function with a initialized to 0 and included in the network parameters Θ to be optimized by gradient descent. Note that with $a = 7$, this recovers a function very similar to $\log(\max(10^{-7}, x))$ (see Figure 3.7, p. 57).

[8] Previous chapters rely on datasets I cannot share, so publishing the code seemed less useful.
[9] Errors reported are the means and standard deviations over five repetitions of each experiment.

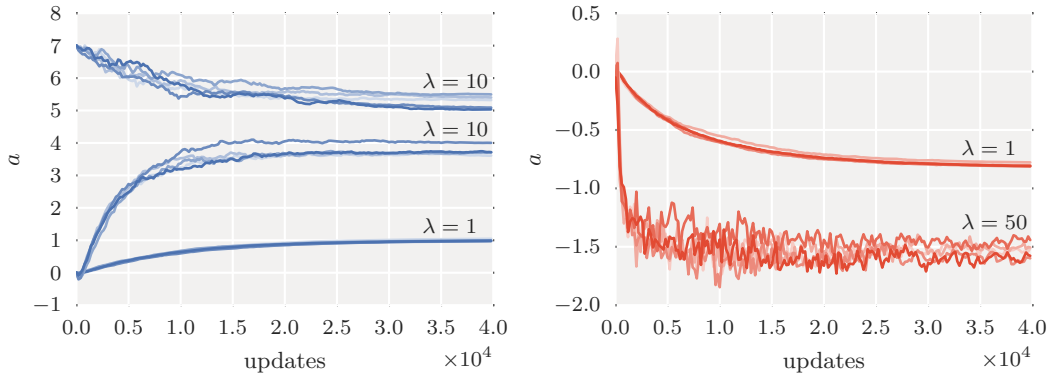


Figure 9.12: Evolution of a while training magnitude transformations $\log(1+10^a \cdot x)$ (*left*) and $x^{\sigma(a)}$ (*right*), with different learning rate factors λ and initial a, and five repetitions per setting.

To ensure the input to the first convolution is always standardized to zero mean and unit variance even if a changes, I employ batch normalization (Ioffe and Szegedy, 2015) after the magnitude transformation (normalizing across the batch and the temporal dimension, so each frequency band is normalized separately, as I usually do as part of the feature extraction).
I trained a network on *Jamendo*, starting from the settings that achieved 7.3% above. As the gradient wrt. a is of a very different magnitude than gradients wrt. other network parameters, I replace Nesterov momentum with ADAM (p. 32), which is invariant to the scale of gradients. This requires changing the learning rate; I set it to 0.001, momentum (β_1 in Eq. 2.41, p. 32) to 0.9 and otherwise follow Section 9.3.3 (including the learning rate decay).
With a fixed to 7, this yields an error of 6.5(±0.3)%, better than 7.3% not due to ADAM, but because the input batch normalization introduces additional variance compared to a fixed input standardization, improving generalization.
Initializing $a = 0$ and learning it ends at $a \approx 1$ and 6.6(±0.3)% error. Boosting the learning rate for a by a factor $\lambda = 50$ ends at $a \approx 4.6$ and 6.7(±0.2)%. Starting at $a = 7$ with boosted learning rate ends nearby at $a \approx 5.1$.
As an alternative magnitude transformation, I tried $x^{\sigma(a)}$, a power function with a learnable exponent in range $(0, 1)$ enforced by the logistic sigmoid. Initializing $a = 0$ (recovering square root magnitudes used by some authors, see p. 57) and training the network ends at $a \approx -0.4$ and 6.8(±0.2)% error. With learning rate boosted by $\lambda = 50$, it ends at $a \approx -1.5$ and 6.7(±0.5)%.
Figure 9.12 shows the development of a over the training time for different settings, with five runs per setting. We can see that the parameter is clearly adjusted, but in the end, everything performs about the same.

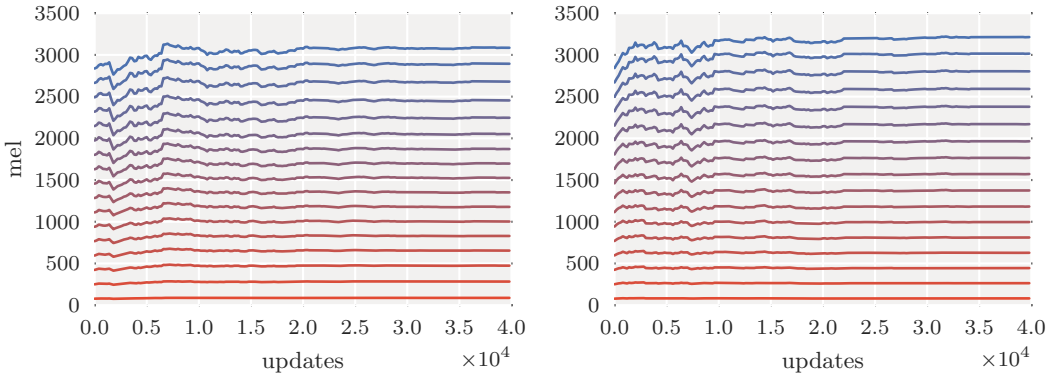


Figure 9.13: Evolution of mel filterbank frequencies over training time, learning the distances between frequencies in mel, with learning rate factor $\lambda = 50$, for two repetitions. For clarity, only 17 of 82 frequencies are shown.

Learned mel filterbank: Just like the magnitude transformation, the mel filterbank parameters are usually chosen by hand. In literature, there have been some attempts to replace the mel filterbank with freely-learned filters (e.g., Cakir et al., 2016), but this discards the spatial layout of the frequency dimension, preventing the use of 2D convolutions on top. This can be circumvented by fixing the frequency range per filter (Sainath et al., 2013a; Qu et al., 2016; Yu et al., 2017), but then these ranges cannot be learned.
An alternative is to formulate the mel filterbank as an operation differentiable wrt. the filterbank parameters, and train those as part of the network. The filterbank output consists of weighted sums of the input frequency bands, and is differentiable wrt. the weights. The weights form triangular filters with defined minimum, centre and maximum frequencies (Figure 3.6, p. 53). Denoting these frequencies as a, b, c, respectively, a filter can be formed as:[10]

$$w(f, a, b, c) = \max\left(0, \min\left(\frac{f-a}{b-a}, \frac{f-c}{b-c}\right)\right), \tag{9.2}$$

where $w(f, a, b, c)$ denotes the weight for frequency f. This constructs the triangular filter as the minimum of two linear ramps (the left and right side) clipped below zero, and is differentiable wrt. a, b, c.[11] For a mel filterbank, adjacent filters are chosen such that the centre and maximum frequency of a filter equal the minimum and centre frequency of the next, respectively (see p. 53). Constraining parameters in the same way, we end up with $M + 2$ learnable positions for a filterbank of M bands.

[10] Compared to Figure 3.6, filters are not scaled to unit area as we standardize the output anyway.
[11] Except at the discontinuities, to be handled as for the rectifier and max-pooling (p. 26).

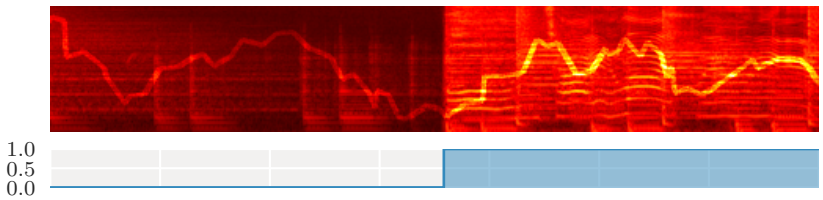
I evaluate this approach in the same way as the learnable magnitude transformation, treating the filter positions as part of the network parameters Θ, using batch normalization to standardize the output despite the changing filterbank, and ADAM to cope with the different gradient magnitudes. For unmodified filterbank learning rates and rates boosted by $\lambda = 50$, errors are 6.7($\pm$0.4)% and 7.2($\pm$0.6)%, respectively, still compared to 6.5($\pm$0.3)% with fixed parameters. Figure 9.13 shows the evolution of the filter positions over training time for a learning rate boosted by $\lambda = 50$, when learning the distances between positions. Again, while the parameters were adjusted during training, it did not result in an improved filterbank.

I tried several variations: Initializing the positions on a linear or mel scale, parameterizing them linearly or via a mel transformation, directly or via their distances, or even learning the minimum/maximum frequency and shape of the mel scale (as $m(f) = 1000/\log(1 + 1000/a) \cdot \log(1 + f/a)$, with $a = 700$ recovering Equation 3.3 and larger/smaller a being more linear/logarithmic). None lead to any improvements in classification error.

In parallel to my experiments, Seki et al. (2017) learned filter positions and bandwidths, replacing triangular filters with Gaussian filters because they wrongly asserted the former are not differentiable. They report slightly improved word error rates, but the filter parameters hardly changed from their initialization (Seki et al., 2017, Fig. 2).

Sloped lines augmentation: In Section 9.8, we found that the networks seemingly learned to detect sloped or wiggly lines in spectrograms, and mistake any such lines for singing voice. In Section 9.6.3, we saw that networks can become more robust to particular variations in the input data by training on modified examples. An obvious idea is to use the same technique to teach the networks to ignore simple wiggly lines, in the hope that this will force the networks to use more refined cues to detect singing voice, and reduce false negatives (such as shown in Figure 9.10) or false positives (such as other instruments capable of bending notes).

To this end, I devised an algorithm producing random sloped lines akin to the hand-drawn examples in Figure 9.11, with randomly chosen line width and opacity, and used it to augment training examples with a chance of 10%. Figure 9.14 shows an augmented test example with singing voice in the second half. While a baseline network predicts singing voice throughout, a network trained with sloped lines augmentation ignores the artificial lines. On Jamendo, it obtains a classification error of 7.5($\pm$0.2)% compared to 7.3($\pm$0.5)% for the baseline (the earlier one without input batch normalization). It seems that instead of finding a better way to detect singing voice, the network merely found a way to specifically ignore the augmentation.



(a) mel spectrogram with sloped lines augmentation and ground truth

(b) singing voice predictions of baseline network

(c) predictions of network trained with sloped lines augmentation

Figure 9.14: Algorithmically created sloped lines (a) are mistaken for singing voice (b) just like the examples in Figure 9.11, but the network can be trained to ignore them by augmenting the training data (c).

Dropout variants: In its original formulation, dropout (p. 38 in Section 2.2.5) randomly masks individual units in a fully-connected layer to promote independence between units. For convolutional layers, the naive approach is to randomly mask individual values in the stack of feature maps. This did not work well in my experiments, so throughout this thesis, I applied dropout to the fully-connected layers of networks only.

An alternative is to drop convolutional units, i.e., mask complete feature maps. This was first proposed by Tompson et al. (2015, Sec. 3.2), termed *spatial dropout*. They argue that if neighbouring input pixels are correlated (as is the case for images and spectrograms), neighbouring pixels in convolutional feature maps are correlated as well, and masking them individually hardly removes any information to enhance independence. They find that dropping full feature maps instead works much better.

For spectrograms, the two spatial dimensions have a different meaning. Another possibility is to drop full frequency bands, across all feature maps – similar to the random frequency filtering augmentation (p. 167), this could encourage the network to become robust to missing frequency bands, but affect both the input and hidden layers.

I trained networks with all three variants, comparing results to the baseline of 6.5(±0.3)%. When dropping 10% of individual input pixels, channels or frequency bands (of all convolutional layers except for the very first), classification error reaches 6.7(±0.1)%, 6.3(±0.3)%, and 6.9(±0.3)%, respectively.

It seems spatial dropout is the only variant that improves results. Following Tompson et al.'s argument, dropping individual bands may be too fine-grained, since neighbouring frequency bands in a spectrogram are correlated. Note that the dropout rate for the convolutional layers is much lower than for the fully-connected layers (10% against 50%) – this is common in literature (e.g., Srivastava et al., 2014, Sec. 6.1.2; Clevert et al., 2016, Sec. 4.2) since convolutional layers have fewer parameters and need less regularization. Increasing the rate to 20% deteriorates results for all three variants.

Stereo input: Many music recordings are mastered such that the main vocalist has the same loudness in the left and right channel of a stereo signal, and thus appears in the centre in common playback configurations. All experiments in this thesis reduce the input to a monophonic signal during preprocessing, to ensure a classifier cannot rely on such stereophonic cues, and has to solve the task in a more robust way – after all, humans do not require binaural input to address the tasks in this thesis either. Still results might be improved by training a network to use these cues when possible.

As a first step towards this, I extended the baseline network to two input channels processing the left and right channel spectrograms, respectively, and trained it on stereo input, expecting that it would learn that singing voice is almost always in the centre and improve results. The next step would have been to augment the training data by randomly downmixing the input (possibly with varying weights for the two channels) and replicating it over the channels, so the network will learn to cope with monophonic input as well. However, for some reason, even the first step deteriorated results.

Better multiple-instance learning: As discussed in Section 9.5.1.2 (p. 170), a natural way to tackle a multiple-instance learning problem is to build a model on the assumption that the label of a bag (here, a song) is the maximum of the labels of its instances (here, the 1.6-second excerpts classified by the CNN). In our case, this means casting the CNN as a fully-convolutional network (Appendix B, p. 205) that produces a time series of predictions for a spectrogram rather than a single prediction for a single excerpt, taking the maximum of these predictions, and computing the cross-entropy loss against the target label for the song. Compared to the naive excerpt-wise training procedure of Section 9.5.2.1 (p. 172), minimizing this loss directly affects the maximum prediction per song only, increasing it for positive examples, and decreasing it for negative examples.

I explored this training procedure for the *In-House* A^+ dataset. Compared to the naively trained network, which has high recall and low precision for individual excerpts (Figure 9.15b, repeated from Figure 9.8d, p. 183), it results in a network with high precision and low recall that only detects a subset of

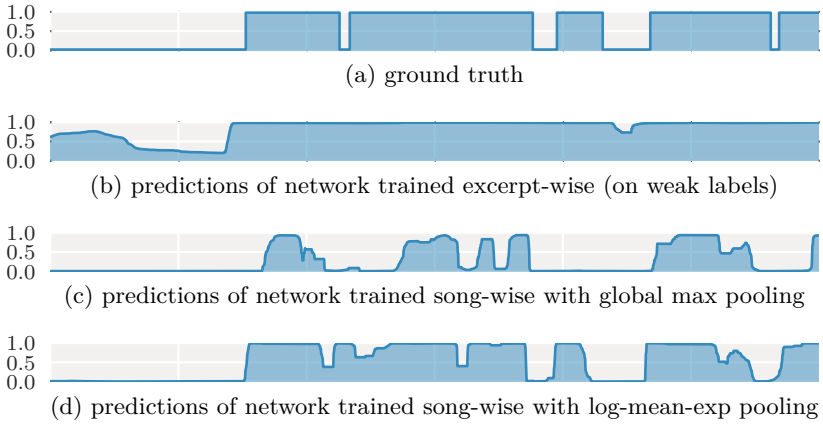


Figure 9.15: Training a network on excerpts naively assigned the song-wise label produces a lot of false positives (b). Training on full songs with global max pooling reduces false positives, which is helpful for predicting song-wise labels, but also reduces recall (c). Global log-mean-exp pooling improves recall, but not enough for subsecond-wise predictions (d). Predictions are shown for the same clip as in Figure 9.8.

vocal excerpts (Figure 9.15c). While this gives excellent song-wise predictions – which I later exploited for the task of detecting bird calls (Grill and Schlüter, 2017) – it is neither suitable for subsecond-wise detection, nor as starting point for the self-improvement recipe of Section 9.5.2.3 (p. 174).
The strong selectivity of max pooling (and the resulting sparse gradient) can be mitigated by pooling with *log-mean-exp* instead:

$$\operatorname{lme}(\boldsymbol{y}; a) = \frac{1}{a} \log \left(\frac{1}{T} \sum_{t=0}^{T-1} \exp(a \cdot y_t) \right) \tag{9.3}$$

Here, $\boldsymbol{y}$ denotes the time series of excerpt-wise predictions $(y_0, y_1, \ldots, y_{T-1})$. Pinheiro and Collobert (2015, Eq. 6) used this as a replacement for global max pooling in a setting very similar to mine, for training a CNN to perform pixel-wise labelling of images given global labels only. The hyperparameter a controls the selectivity: with large a, the result mostly depends on the largest y_t, approximating max pooling, while a small a approximates mean pooling. Figure 9.15d shows predictions for a network trained with log-mean-exp pooling and $a = 1$. While this noticeably improves recall compared to max pooling, it still cannot compete with the three-step self-improvement recipe.

9.9 Discussion

Musical data augmentation: In the first part of this chapter, I evaluated seven label-preserving audio transformations for their utility as data augmentation methods on music data, using singing voice detection as the benchmark task. Results were mixed: Pitch shifting and random frequency filters brought a considerable improvement, time stretching did not change a lot, but did not seem harmful either, loudness changes were ineffective and the remaining methods even reduced accuracy. I expect that data augmentation would prove beneficial for a range of other music understanding tasks, especially those operating on a low level.

The strong influence of augmentation by pitch shifting, both in training and at test-time, indicates that it would be worthwhile to design the classifier to be more robust to pitch shifting in the first place. The architecture for the second series of experiments in this chapter already follows up on this idea, but leaves ample room for improvement.

Frequency filtering as the second best method deserves closer attention. The scheme devised for the experiments is just one of many possibilities, and probably far from optimal. A closer investigation of why it helped might lead to more effective schemes. For example, it is imaginable that narrow-band filters removing frequency components at random would force a classifier to always take all harmonics into account, and become robust to masked partials – however, as we saw in the previous section, filters should not be so narrow that they remove a single spectral band only.

Learning from weak labels: As the second contribution of this chapter, I explored how to train CNNs for singing voice detection on coarsely annotated training data and still obtain temporally accurate predictions, closely matching performance of a network trained on finely annotated data. I expect the recipe to carry over from human voice to other musical instruments, if good contrasting examples are available – training on weakly-annotated data can only learn to distinguish instruments that occur independently from one another in different music pieces. However, singing voice may be easier to learn to detect than other instruments due to the way it is commonly used: in the training clips of the *In-House A* dataset that contain vocals, they are featured for 70% of the running time (between 13% and 99% per clip), and recordings are often mastered to make vocals the most prominent instrument in a song.

Furthermore, I demonstrated a method for localizing the spectral bins that contain singing voice, without requiring corresponding ground truth for training. This could be a starting point for instrument-specific source separation, but also used to visualize and auralize precisely which content in a spectrogram was responsible for a particular false positive given by a network, and thus give a hint on how to enrich the training data to improve results.

Comparison: Both musical data augmentation and learning from weakly-labelled data can be used to reduce annotation efforts – the former because it permits smaller training sets, the latter because it simplifies the annotation task. My hypothesis was that even with data augmentation, a large dataset of song-wise annotations would be superior to a small dataset of temporally accurate labels, and possibly even less expensive to create. However, in this chapter, the networks trained on 9663 weakly-labelled clips performed on par with networks trained on 100 finely-labelled clips. Moreover, from the numbers given in Section 9.6.1 (p. 175), we can infer that the former took about 24 h to annotate, while the latter took 3.5 h. Finally, while the networks trained on weakly-annotated data produce good prediction curves, when used for binary classification, we still need to choose a suitable threshold (e.g., optimizing accuracy, or a precision/recall tradeoff), and I did not find good heuristics for selecting such a threshold solely based on weakly-labelled examples.

Thus, it seems that a small and accurate dataset is advantageous over a large inaccurate dataset. However, it is helpful to know that training from weakly-labelled clips can give high temporal accuracy *at all*, and is available as an option if such weak labels are easy to obtain. For future work, it would be interesting to compare the two methods on more even grounds. Specifically, one could investigate if weak labelling, fine labelling or a combination of both provides the best value for a given budget of annotator time.

Singing voice detection: Regarding the task of singing voice detection, I advanced the state of the art from 7.7% to 7.3% classification error on the public *RWC* dataset, and from 8.5% to 7.7% on the *Jamendo* dataset (Table 9.2) using train-time and test-time data augmentation, and further reduced the error on *Jamendo* to 6.3% with incremental improvements of the architecture and training procedure (Section 9.8). Better solutions could be reached by training larger CNNs or bagging multiple networks (indeed, averaging the predictions of five networks reaching 6.3(±0.3)% individually further reduces the error to 5.9%, and blindly bagging all networks of the previous section reaches 5.8%).

But despite their state-of-the-art performance, examining the networks showed that they mostly learned to detect sloped lines in the input spectrograms (Section 9.7, p. 184). This strategy is also followed by hand-designed approaches: For example, the voice detector of Sonnleitner et al. (2012) is based on comparing neighbouring spectral frames to determine whether one is a slight transposition of the other, effectively finding sloped harmonic tones, and KAML (Liutkus et al., 2015) assumes that vocals are more irregular than other instruments. The fact that such an approach gives state-of-the-art results is to be blamed on the datasets commonly used for singing voice detection: Both *Jamendo* and *RWC* exclusively consist of songs containing vocals, leaving little room for vocal-like instruments to be mistaken for singing voice. Further improvements in singing voice detection will

require datasets that explicitly challenge the assumption that all sloped lines are vocals, possibly with plentiful instrumental pieces featuring saxophone, violin, or electric guitars with bended notes, both for training and for evaluation.

10 Conclusion

To wrap up this thesis, I will discuss the work presented in the previous chapters as a whole, and provide an outlook on possible avenues of future research. For a detailed summary of the achievements of this thesis, please refer to Section 1.4 (p. 4) instead, and for task-specific discussions, see the last section of each chapter.

10.1 Discussion

In Chapters 5 to 9, I applied deep learning methods to five different music understanding tasks: music/speech detection, fast music similarity estimation, onset detection, music boundary detection, and singing voice detection. In all five tasks, results improved over the previous state of the art. Accomplishing this in the course of a single PhD thesis was only feasible with deep learning: With manually-constructed algorithms or with hand-designed features and more classical machine learning techniques, it would have required a lot of detailed domain and task knowledge to improve over the state of the art even for a single task. Of course, successfully applying deep learning to audio signals requires a nontrivial amount of background knowledge as well (Chapters 2 to 4), and tuning it to outperform existing results may require a lot of experimentation,[1] but most of this expertise is reusable across a wide range of tasks.

For both sequence labelling and event detection tasks (i.e., all tasks except for accelerating music similarity measures), my approaches share a common modelling assumption: each prediction depends on a limited local input context only. For example, the singing voice detector assumes that whether somebody is singing at exactly 30 seconds into a music piece does not depend on the audio signal at 5 or 40 seconds into the piece, but only on 800 ms before and after the decision point.

[1] For example, finding a way to beat Dominik Schnitzer's approach to speeding up Gaussian-based music similarity measures with neural networks was not easy (Section 6.6.3, p. 107). On the other hand, it only took a week to learn Theano, implement my first CNN and write a paper improving upon Sebastian Böck's approach to onset detection (Schlüter and Böck, 2013).

This assumption is clearly violated in practice: Music pieces have short-term and long-term temporal structure. One way to model this dependency would be using Recurrent Neural Networks (RNNs). However, as already discussed in Section 7.7 (p. 135) for the case of onset detection, treating excerpts as independent makes it much easier to train a model, and for the tasks considered in this thesis, the temporal structure of most music pieces is not strong enough to make accurate predictions based on past or future information even just a couple of seconds away anyway (with the possible exception of extremely repetitive music such as minimal techno). Short-term correlations for sequence labelling – e.g., the assumption that the presence of singing voice will most often stay unchanged from one time step to the next – are more easily incorporated by temporal smoothing of the network predictions with a median filter.

A common criticism of neural networks compared to hand-designed approaches is that the former are black boxes: While they may appear to solve the task they were trained for, it is unclear how they produce their predictions, making it difficult to assess whether their solution is reliable. In this work, I tried to address this by examining networks after training, visualizing the features and internal states, computing saliency maps, or probing them with customized input (see the previous to last section in most chapters). In all cases, my findings indicate that the networks exploit the same or similar ideas as existing hand-designed methods, but use a wide array of minor variations of each idea with all parameters fine-tuned to optimize performance. While my examination was not at all exhaustive, it demonstrates that neural networks need not be treated as black boxes, but it also shows that their advantage lies in a feature out of reach for hand-designed algorithms. Still, for some applications, full control of the algorithm is more important than performance on a set of benchmark datasets, ruling out complex neural networks.

Another common criticism is that neural networks are computationally more expensive than hand-designed approaches, so even if the latter give worse results, they are allegedly suited better for real-time or embedded applications. The network architectures in this thesis were not designed with efficiency in mind, and may not be competitive with hand-designed algorithms in this respect (e.g., the SuperFlux algorithm of Böck and Widmer (2013) provides onset detections with much fewer operations than my CNN). However, neural networks can easily be made smaller to also be computationally cheaper and give worse results: either by simply reducing their width (number of units per layer) and depth (number of layers), or with more advanced methods for compression (e.g., Hinton et al., 2014; Han et al., 2016; Garipov et al., 2016; Ullrich et al., 2017) and complexity reduction (e.g., Gupta et al., 2015; Rastegari et al., 2016). Of course, it is unclear whether they will reach a better or worse operating point than a given hand-designed method, and it would be very interesting to perform a comparison on even grounds with matching computational budgets.

10.2 Outlook

While my results improved over the state of the art for all five tasks considered in this thesis, and by now, deep learning has become more and more widespread in the music information retrieval community, there are plenty of opportunities for further improvements.

An obvious way is to curate larger annotated datasets such as the Million Song Dataset[2] by Bertin-Mahieux et al. (2011) with extensions for auto-tagging, similarity estimation, cover song detection (Bertin-Mahieux and Ellis, 2011) and aligned MIDI files providing a host of other information (Raffel, 2016), or MusicNet by Thickstun et al. (2017) for instrument detection and transcription. For example, the observed deficiencies in singing voice detection (Section 9.8, p. 187) are probably primarily a symptom of the datasets rather than the method, and might be most easily addressable by systematically extending the datasets with more instrumental examples.

On a more technical level, the network architectures used in my work are not thoroughly optimized for their tasks and can probably be tuned further. More generally, they could profit from recently proposed building blocks and network designs not explored in this thesis, such as Inception modules (Szegedy et al., 2015, 2016a,b; Chollet, 2016), residual connections (He et al., 2016a,b), DenseNets (Huang et al., 2017), or self-normalizing neural networks (Klambauer et al., 2017). Dilated convolutions (Yu and Koltun, 2016, also see Appendix B.3) could be especially interesting for processing music recordings: Apart from providing a way to process a larger temporal context, using different dilation factors in parallel on spectrograms could allow to detect regular patterns over time or harmonic overtones over frequency.

Another line of research would be to further explore training with constrained annotation budgets, following up on the work of Chapter 9. Instead of starting with a labelled dataset and finding the optimal way to predict annotations from inputs, it would start with unlabelled data and either research how to spend a given budget of annotation time to obtain the best possible prediction model for a particular task, or research how to minimize annotation time to obtain predictions of a particular quality. This would entail combining strongly supervised learning with unsupervised learning, learning from weakly-labelled data and active learning, and answer important questions for practical applications.

Finally, the most promising avenue for future work improving the performance on music understanding tasks is to employ multi-task learning (Caruana, 1997): Instead of designing and training models for different tasks in isolation, train a joint model that solves multiple related tasks at once to leverage commonalities.

[2] Unfortunately, it lacks the audio data required for the approaches discussed in this thesis, but for some tasks, it may suffice to acquire preview snippets from a music distribution service.

The inherent structure of music results in plentiful interrelations between aspects often tackled by separate models: for example, onsets are temporally related to beats and downbeats, downbeats to chords and musical boundaries, chords are related to keys and notes, genre and mood to instrumentation and tempo. While such dependencies between tasks have been recognized and exploited manually (e.g., Zenz and Rauber, 2007; Mauch et al., 2009) or with transfer learning (e.g., Hamel and Eck, 2010; van den Oord et al., 2014; Hamel et al., 2013), only few works on automated music understanding actually train a single model to address multiple tasks jointly (e.g., Papadopoulos and Peeters, 2011; Weston et al., 2011). Doing so allows a model to develop shared processing steps or representations informed by multiple tasks, and multiple datasets. Even for closely tied tasks, it provides a way to expand the amount of training data: a network trained to predict a song-wise singing voice label, subsecond-wise singing voice labels and a soft mask for singing voice extraction could leverage weakly-annotated, finely-annotated and multi-track training sets at once. Multi-task learning could bring music understanding systems closer to how humans understand music and improve results over solving tasks in isolation. At the same time, automated music understanding could serve as a comprehensive and varied test bed for improving multi-task learning.

Appendix A

Commercial Applications

Some outcomes of this thesis – or of work closely related to it – have been licensed to industry to solve a variety of problems. In this appendix, I will briefly describe three examples to demonstrate the practical applicability of the research discussed in the previous chapters.

A.1 Royalty Collection

In 2012, *SWISSPERFORM*, the collecting society for neighbouring rights in Switzerland, approached us with a problem. The royalties they charged radio stations were based on airtime reports given by the radio stations themselves. SWISSPERFORM had the suspicion that these reports underestimated the amount of music aired, and aimed for a more objective and fairer estimate. Manually annotating all stations in question was infeasible, so they were seeking an automated or semi-automated approach for measuring the amount of music aired by a station. As an additional requirement, the method had to distinguish between music played in the foreground and music overlaid by speech, since the royalties to be paid for the latter are lower.

In collaboration with Reinhard Sonnleitner and Gerhard Widmer, we recorded 30-minute excerpts from six Swiss radio stations randomly distributed over the course of a week, a total of 30 hours, had them annotate by students via a custom web interface with respect to the presence of speech and/or music, and set out to find a way to automatically determine the amount of speech, music, and speech with background music: Reinhard Sonnleitner by designing a feature for speech detection and training a random forest on top (Sonnleitner et al., 2012), to be combined with an existing music detector by Seyerlehner et al. (2007), and me using deep neural networks (Chapter 5 or Schlüter and Sonnleitner, 2012).

We recorded and annotated excerpts from two more weeks to validate our approaches, and compiled a report on the estimated amount of music aired by the stations. In addition, we licensed the speech and music detectors by Sonnleitner and Seyerlehner to SWISSPERFORM – while my approach was both faster and a little more accurate on our validation data, deep learning was deemed a too experimental and poorly-understood technique at the time.

A.2 Radio Broadcast Monitoring

Via our publication on speech and music detection (Schlüter and Sonnleitner, 2012), the Danish company *RadioAnalyzer* found and contacted us in 2013 for help solving a slightly different problem: Detecting from when to when a radio station is airing a full music piece, to improve temporally inaccurate playlists they obtained from external sources, and thus improve results for their automatic radio broadcast monitoring application.

With access to recordings of all songs possibly aired by each station, this could be solved by matching the broadcast recordings to the references – simplified by the fact that coarse playlists are known – and determining the precise beginning and ending times of each match. However, except for monitoring stations with a limited repertoire closely following the music charts, this is infeasible for small to medium-sized companies. For a more general solution, RadioAnalyzer's idea was to use our music detector to find long enough sequences of music to qualify as airings of full songs. But since the detector is designed to also detect background music, this would not work – some stations play music throughout, even during the news.

The solution I developed in collaboration with Thomas Grill instead focuses on the speech detector.[1] Since radio broadcast recordings are predominantly composed of music and speech, it serves as a proxy for finding long unobstructed passages of music. However, the detector can mistake rap parts in a music piece for speech, and it cannot detect the transitions between songs if there is no intermittent announcement by the radio host or a jingle containing speech. Both issues are solved by combining it with a segmentation algorithm by Thomas Grill – a hand-designed precursor to the CNN-based approach of Chapter 8 – tuned to find song transitions. With some heuristics, it allows us to ignore spurious speech detections in the middle of a song, to find the transition point between two songs played back to back (with more precise timing if it is accompanied by a jingle caught by the speech detector), and to obtain precise beginning and ending times (by detecting the radio host announcements or advertisements). RadioAnalyzer successfully uses our software in production since 2014.

A.3 Music Recommendation

Music recommendation is the task of suggesting music pieces (or albums, or artists) to a user that she or he might want to listen to, but possibly has never heard before, based on cues about the user's preferences such as her or his listening history.

A solution commonly employed by large online music distributors or streaming services relies on *collaborative filtering*: Compare the user's preference cues (e.g.,

[1] This time, we used the one based on neural networks described in Chapter 5.

listening counts or purchase history) to those of others to find similar users, then recommend items preferred by many such users that are not yet known to the user requesting a recommendation. However, this can only ever recommend items that other users have already built a preference for, and it is biased towards recommending popular songs, making those even more popular.

An alternative is to follow a content-based approach: Instead of comparing users, compare the items, and recommend those that are similar to items preferred by the user in question. This avoids the popularity bias of collaborative filtering, but requires a way to compare items. Some services achieve this by having experts provide detailed annotations of items and comparing these annotations, but this does not easily scale to large catalogues, and can only recommend items that have already been annotated. This is one of the potential applications of music similarity measures: By providing an automated way for a content-based comparison of music recordings, it enables music recommendations undisturbed by a popularity bias, across large collections.

For his dissertation, Schnitzer (2011) developed a way to accelerate the music similarity measure of Pohle et al. (2009) to be applicable to a few million items. He also implemented it efficiently in C/C++ and created an open source music similarity library[2] it could be embedded into as a plugin to be licensed from OFAI. For Chapter 6, I developed a way to accelerate the measure of Seyerlehner et al. (2010b) to scale to a few tens of millions of items. Subsequently, I improved the existing C/C++ implementation of the similarity measure by Reinhard Sonnleitner, added the best-performing indexing method of Section 6.6.2 and the compression approach of Section 6.7, and also turned it into a plugin for the software library to be licensed from OFAI. Both measures have been licensed and put to use, with the version from my thesis being especially useful for large collections, or for comparing song segments instead of songs (which increases the number of items, but is helpful for production music that changes its musical qualities over the course of a piece).

[2] `http://musly.org`, accessed June 2017

Appendix B

Efficient CNN Predictions on Time Series

Both for event detection and sequence labelling, I train CNNs to solve a local classification task: Given a short, fixed-size input sequence (in my case, a spectrogram excerpt), output a single number in $[0, 1]$ indicating the probability of an event or the probability of one of the classes being present at the temporal centre of the sequence. At test time, we are given an input sequence of unknown length, and are expected to produce the locations of all events, or a class label for every time step. In this appendix, I will briefly explain the naive solution and a more efficient way to accomplish this. While helpful to know, this is neither essential enough to warrant inclusion in Chapter 2, nor trivial enough to be explained within the main chapters of the thesis.

B.1 Naive Approach

Figure B.1 depicts an example CNN architecture processing an input spectrogram excerpt of 31×80 (31 frames of 80 bands) with a convolutional layer of 16 filters of shape 5×3, a max-pooling layer of 1×6, a fully-connected layer of 32 units and another of a single output unit. That is, it implements a function $f : \mathbb{R}^{31 \times 80} \to \mathbb{R}$.

When given a spectrogram of 100×80 at test time, we need to compute a time series of 100 predictions – one per frame – and apply a peak-picking algorithm for event detection, or a threshold for sequence labelling.

To obtain these 100 predictions, the naive solution is to pad the spectrogram with 15 frames on either side,[1] cut out all $15 + 100 + 15 - 31 + 1 = 100$ possible excerpts of 31 consecutive frames, pass each excerpt to the CNN, and assemble the results in chronological order:

$$p_t = f(\boldsymbol{X}_{t:t+31}), \tag{B.1}$$

where t is the time step, f is the CNN, and $\boldsymbol{X}$ is the padded spectrogram.

In the following, we will discuss how to obtain the same result more efficiently.

[1] Padding should be done with content that does not adversely affect the predictions, or in a way that was already used in training, but this is not the focus of this appendix.

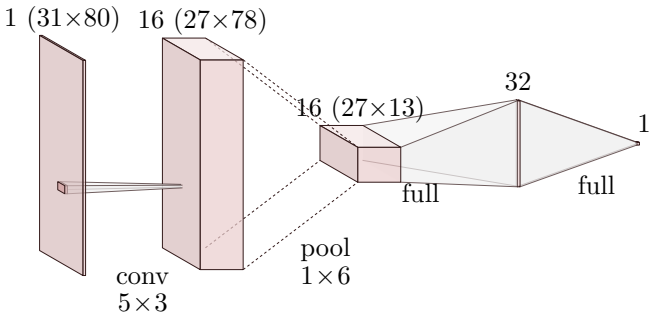


Figure B.1: A small convolutional neural network used as an example.

B.2 Fully-Convolutional Network

Our goal is to reformulate the network to obtain a function $g : \mathbb{R}^{m\times 80} \to \mathbb{R}^m$ that is more efficient than applying f to overlapping input excerpts. The first step towards this is to realize that the naive approach performs a lot of redundant computations. The first layer convolves the input with filters of 5×3, which is a shift-equivariant operation: Translating the input by one time step also translates the output by one time step. Furthermore, the operation is local: Computing one frame of the output only requires processing 5 input frames. Taken together, when the output of the convolutional layer for $\boldsymbol{X}_{t:t+31}$ is already known, the output for $\boldsymbol{X}_{t+1:t+32}$ only requires computing a single new frame from 5 input frames.

A straightforward way to exploit this redundancy is to change the order of operations. Instead of cropping out spectrogram excerpts of 31 frames and processing them with the convolutional layer, we can process the complete padded spectrogram of 130 frames with the convolutional layer and crop out excerpts of 27 frames from the result (which has $130 - 5 + 1 = 126$ frames then).

The same arguments and the same solution apply to the max-pooling layer: Instead of separately processing 27-frame excerpts from the convolutional layer, we can process all 126 frames at once and produce excerpts from the result.

For the fully-connected layers, there is no redundancy we can exploit. However, we can reformulate it for notational convenience. Note that the first fully-connected layer computes a dot product of 16 excerpts (the number of channels produced by the convolutional layer, and kept by the max-pooling layer) of 27 frames and $(80-3+1)/6 = 13$ bands by a 5616×32 weight matrix. Passing overlapping excerpts to this layer is exactly equivalent to a convolutional layer of 32 filter sets of shape 27×13 (with each filter set processing the 16 input channels): Instead of producing 100 vectors of length 32 by processing 100 excerpts with the fully-connected layer,

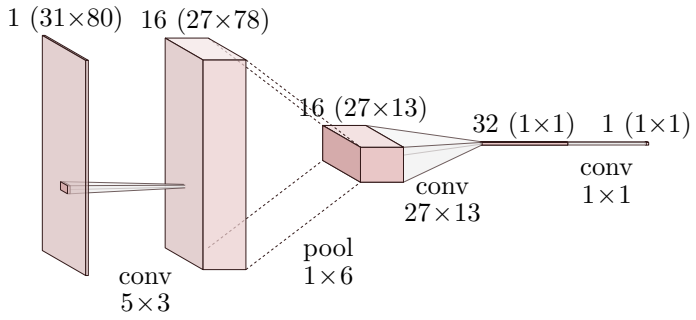


Figure B.2: The network of Figure B.1 cast as a fully-convolutional network.

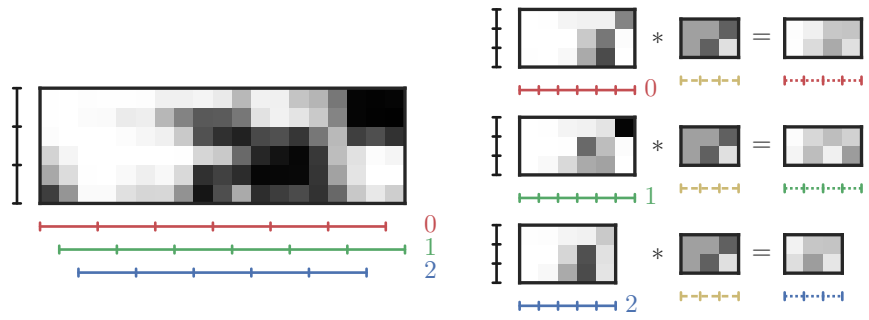
we can produce 32 feature maps of 100×1 with the convolutional layer. Similarly, the output layer can be replaced by a convolutional layer of a single filter set of shape 1×1.

The resulting network is called a Fully-Convolutional Network (FCN), since it only employs convolutional and pooling layers. It can process any spectrogram of $m \geq 31$ frames into an output of $m - 30$ time steps. Combined with spectrogram padding, it yields the function g we asked for. It is depicted in Figure B.2.

B.3 Handling Temporal Pooling

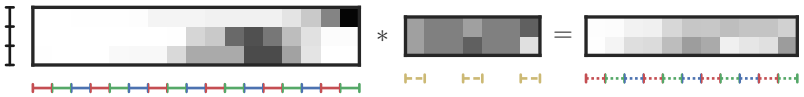
Note that the example CNN we discussed so far is a special case: It only employs max-pooling over features, not over time. If we replace the 1×6 max-pooling with 3×6 max-pooling, the first fully-connected layer will process input excerpts of 9×13 (instead of 27×13). We can still turn it into a fully-convolutional network, with a convolution of 9×13 filters replacing the first fully-connected layer, but due to the temporal pooling, applying it as is would give a prediction for every third frame only. In particular, for the padded input spectrogram of 130 frames, it would produce an output of $\lfloor (130 - 5 + 1)/3 \rfloor - 9 + 1 = 34$ frames instead of the 100 frames obtained with the naive approach.

A simple way to obtain predictions at the full resolution is to pass the input three times, with zero, one and two beginning frames omitted, respectively, and interleave the results. To avoid redundant computation in the first convolutional layer, it can still be applied upfront, before forming the cropped versions passed to the remaining layers (Giusti et al., 2013; Sermanet et al., 2014, Sec. 3.3). Figure B.3 illustrates this operation for some toy data processed by 3×2 max-pooling followed by a 3×2 convolution: Passing three cropped versions through max-pooling and convolution creates low-resolution results we can interleave to obtain full-resolution outputs.



(a) input, with indicated pooling grids at three temporal offsets

(b) input pooled with three offsets and convolved with a filter



(c) input with overlapped pooling convolved with the filter dilated by a factor of three

(d) result

Figure B.3: When a CNN processing a 9×6 input with 3×2 max-pooling and 3×2 convolution is directly applied to larger input (a), it will produce an output of reduced temporal resolution (b, upmost row). To efficiently compute the output for every 9×6 input excerpt, we can apply max-pooling with an offset of 0, 1, and 2 frames, convolve separately (b) and interleave the results (d). Equivalently, we can apply overlapping max-pooling and convolve with a dilated filter of two zeros between every filter column (c). The same techniques apply within a larger FCN.

For a network of multiple pooling layers, we can form additional cropped versions before every temporal pooling operation, and interleave all results before the first fully-connected layer, or in the very end.

An equivalent, but conceptually more elegant way is to use max-pooling with a temporal hop size of a single frame – i.e., with pooling windows maximally overlapping in time – and *dilated* convolutions for the following layers with two zeros between every column of their filters (Chen et al., 2015, Sec. 3.1; Sercu and Goel, 2016, Secs. 2–3).[2] This way, the convolution handles both subsampling the input and interleaving the results, as illustrated in Figure B.3c–d. Its efficiency hinges on implementations of dilated convolution and dilated pooling that do not explicitly multiply by the additional zeros. For a network of multiple pooling layers, every pooling layer multiplies the dilation factor to use for the following layers.

[2] On a side note, dilated convolutions also provide a way to expand the receptive field of a CNN at low computational costs and with few learnable parameters (Yu and Koltun, 2016; van den Oord et al., 2016a). I did not explore this in my work.

Bibliography

O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *Proceedings of the 37th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4277–4280, Kyoto, Japan, Mar. 2012. doi:10.1109/ICASSP.2012.6288864. URL http://www.cs.toronto.edu/~asamir/papers/icassp12_cnn.pdf.

R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL http://arxiv.org/abs/1605.02688. Project URL http://github.com/Theano/Theano.

D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. C. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu. Deep

Speech 2: End-to-end speech recognition in english and mandarin. *arXiv e-prints*, abs/1512.02595, 2015. URL http://arxiv.org/abs/1512.02595.

J. Amores. Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence*, 201:81–105, 2013. doi:10.1016/j.artint.2013.06.003. URL http://refbase.cvc.uab.es/files/Amo2013.pdf.

G. An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674, Apr. 1996. doi:10.1162/neco.1996.8.3.643.

S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 577–584. MIT Press, 2003. URL http://papers.nips.cc/paper/2232-support-vector-machines-for-multiple-instance-learning.

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv e-prints*, abs/1611.01491, 2016. URL http://arxiv.org/abs/1611.01491.

V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: A method for efficient approximate similarity rankings. In *Proceedings of the 17th IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 268–275, Washington, D.C., USA, June 2004. doi:10.1109/CVPR.2004.1315173. URL http://vlm1.uta.edu/~athitsos/publications/athitsos_cvpr2004.pdf.

J.-J. Aucouturier and M. Sandler. Segmentation of musical signals using hidden Markov models. In *Proceedings of the 110th Convention of the Audio Engineering Society (AES)*, Amsterdam, Netherlands, May 2001. URL http://csl.sony.fr/downloads/papers/2001/aucouturier-aes2001.pdf.

D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012. URL http://www.cs.ucl.ac.uk/staff/d.barber/brml/.

J. P. Bello, C. Duxbury, M. Davies, and M. Sandler. On the use of phase and energy for musical onset detection in the complex domain. *IEEE Signal Processing Letters*, 11(6):553–556, June 2004. doi:10.1109/LSP.2004.827951. URL http://www.eecs.qmul.ac.uk/former/people/jbc/Documents/Bello-SPL-2004.pdf.

J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035–1047, Sept. 2005. doi:10.1109/TSA.2005.851998.

URL http://www.eecs.qmul.ac.uk/former/people/jbc/Documents/Bello-TSAP-2005.pdf.

Y. Bengio and O. Delalleau. On the expressive power of deep architectures. In *Proceedings of the 22nd International Conference on Algorithmic Learning Theory (ALT)*, pages 18–36, Espoo, Finland, 2011. URL http://www.iro.umontreal.ca/~lisa/pointeurs/ALT2011.pdf.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In P. B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007. URL http://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.

Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. In *Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8624–8628, Vancouver, Canada, May 2013. doi:10.1109/ICASSP.2013.6639349. Preprint http://arxiv.org/abs/1212.0901.

J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, Sept. 1975. doi:10.1145/361002.361007.

A. L. Berenzweig and D. P. W. Ellis. Locating singing voice segments within music signals. In *IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 119–122, New Paltz, NY, USA, Oct. 2001. doi:10.1109/ASPAA.2001.969557. URL http://labrosa.ee.columbia.edu/~dpwe/pubs/waspaa01-singing.pdf.

T. Bertin-Mahieux and D. P. W. Ellis. Large-scale cover song recognition using hashed chroma landmarks. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 117–120, New Paltz, NY, USA, Oct. 2011. doi:10.1109/ASPAA.2011.6082307. URL http://www.ee.columbia.edu/~dpwe/pubs/BertE11-hashedchroma.pdf.

T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 591–596, Miami, FL, USA, Oct. 2011. URL http://ismir2011.ismir.net/papers/OS6-1.pdf.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 11th edition, 2006. ISBN 0387310738.

R. Bittner, J. Salamon, M. Tierney, M. Mauch, C. Cannam, and J. P. Bello. MedleyDB: A multitrack dataset for annotation-intensive MIR research. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, pages 155–160, Taipei, Taiwan, Oct. 2014. URL http://www.terasoft.com.tw/conf/ismir2014/proceedings/T028_322_Paper.pdf.

S. Böck and G. Widmer. Maximum filter vibrato suppression for onset detection. In *Proceedings of the 16th International Conference on Digital Audio Effects (DAFx)*, Maynooth, Ireland, Sept. 2013. URL http://dafx13.nuim.ie/papers/09.dafx2013_submission_12.pdf.

S. Böck, A. Arzt, F. Krebs, and M. Schedl. Online real-time onset detection with recurrent neural networks. In *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx)*, York, UK, Sept. 2012. URL http://www.dafx12.york.ac.uk/papers/dafx12_submission_4.pdf.

S. Böck, F. Krebs, and M. Schedl. Evaluating the online capabilities of onset detection methods. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR)*, pages 49–54, Porto, Portugal, Oct. 2012. URL http://ismir2012.ismir.net/event/papers/049-ismir-2012.pdf.

S. Böck, F. Korzeniowski, J. Schlüter, F. Krebs, and G. Widmer. madmom: a new python audio and music signal processing library. In *Proceedings of the 24th ACM International Conference on Multimedia (ACMMM)*, pages 1174–1178, Amsterdam, Netherlands, Oct. 2016. doi:10.1145/2964284.2973795. URL http://www.cp.jku.at/research/papers/Boeck_etal_ACMMM_2016.pdf.

B. Bogert, M. Healy, and J. Tukey. The quefrency alanysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. In M. Rosenblatt, editor, *Proceedings of the Symposium on Time Series Analysis*, pages 209–243, New York, 1963. John Wiley and Sons.

N. Boulanger-Lewandowski. *Modeling High-Dimensional Audio Sequences with Recurrent Neural Networks*. PhD thesis, University of Montréal, Montréal, Canada, Apr. 2014. doi:1866/11181. URL http://www-etud.iro.umontreal.ca/~boulanni/NicolasBoulangerLewandowski_thesis.pdf.

S. Böck. *Event Detection in Musical Audio: Beyond simple feature design*. PhD thesis, Johannes-Kepler-University, Linz, Austria, Nov. 2016. URL http://www.cp.jku.at/research/papers/Boeck_dissertation.pdf.

R. Cai, C. Zhang, L. Zhang, and W.-Y. Ma. Scalable music recommendation by search. In *Proceedings of the 15th ACM International Conference on Multimedia (ACMMM)*, pages 1065–1074, Augsburg, Germany,

2007. URL http://research.microsoft.com/en-us/um/people/leizhang/Paper/ACMMM07-Cai.pdf.

E. Cakir, E. C. Ozan, and T. Virtanen. Filterbank learning for deep neural network based polyphonic sound event detection. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 3399–3406, Vancouver, Canada, July 2016. doi:10.1109/IJCNN.2016.7727634. URL http://www.cs.tut.fi/~cakir/publications/filterbank_learning_ijcnn_2016.pdf.

A. Camacho and J. G. Harris. A pitch estimation algorithm based on the smooth harmonic average peak-to-valley envelope. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3940–3943, May 2007. doi:10.1109/ISCAS.2007.378662.

M. J. Carey, E. S. Parris, and H. Lloyd-Thomas. A comparison of features for speech, music discrimination. In *Proceedings of the 24th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 149–152, Phoenix, AZ, USA, Mar. 1999. doi:10.1109/ICASSP.1999.758084.

R. Caruana. *Multitask Learning*. PhD thesis, Carnegie Mellon University, 1997. URL http://reports-archive.adm.cs.cmu.edu/anon/1997/CMU-CS-97-203.pdf.

R. Caruana, S. Lawrence, and C. L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 402–408. MIT Press, 2001. URL http://papers.nips.cc/paper/1895-overfitting-in-neural-nets-backpropagation-conjugate-gradient-and-early-stopping.

A.-L. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte Rendu des Séances de L'Académie des Sciences XXV*, Série A (25):536–538, Oct. 1847.

P. Chandna, M. Miron, J. Janer, and E. Gómez. Monoaural audio source separation using deep convolutional neural networks. In *Proceedings of the 13th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 258–266, Grenoble, France, Feb. 2017. doi:10.1007/978-3-319-53547-0_25. URL http://mtg.upf.edu/system/files/publications/monoaural-audio-source_0.pdf.

E. Chávez, K. Figueroa, and G. Navarro. Proximity searching in high dimensional spaces with a proximity preserving order. In *Proceedings of the 4th Mexican International Conference on Artificial Intelligence (MICAI)*, pages 405–414,

2005. doi:10.1007/11579427_41. URL http://www.dcc.uchile.cl/~gnavarro/ps/micai05.pdf.

L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015. URL http://arxiv.org/abs/1412.7062.

K. Cho, A. Ilin, and T. Raiko. Improved learning of gaussian-bernoulli restricted boltzmann machines. In *Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN)*, pages 10–17, Espoo, Finland, 2011. doi:10.1007/978-3-642-21735-7_2. URL http://research.ics.aalto.fi/bayes/papers/files/icann11.pdf.

F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv e-prints*, abs/1610.02357, 2016. URL http://arxiv.org/abs/1610.02357.

D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016. URL http://arxiv.org/abs/1511.07289.

N. Collins. Using a pitch detector for onset detection. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 100–106, London, UK, Sept. 2005. URL http://ismir2005.ismir.net/proceedings/1008.pdf.

J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215–242, 1958. ISSN 00359246. URL http://www.jstor.org/stable/2983890.

X. Cui, V. Goel, and B. Kingsbury. Data augmentation for deep neural network acoustic modeling. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5582–5586, Florence, Italy, May 2014. doi:10.1109/ICASSP.2014.6854671. URL http://www.redes.unb.br/lasp/files/events/ICASSP2014/papers/p5619-cui.pdf.

G. E. Dahl, M. Ranzato, A. rahman Mohamed, and G. E. Hinton. Phone recognition with the mean-covariance restricted Boltzmann machine. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances*

in Neural Information Processing Systems 23, pages 469–477. Curran Associates, Inc., 2010. URL http://papers.nips.cc/paper/4169-phone-recognition-with-the-mean-covariance-restricted-boltzmann-machine.

S. B. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366, Aug. 1980. doi:10.1109/TASSP.1980.1163420.

S. Dieleman. *Learning Feature Hierarchies for Musical Audio Signals*. PhD thesis, Ghent University, Ghent, Belgium, Jan. 2016. doi:1854/LU-8174817. URL http://www.dropbox.com/s/22bqmco45179t7z/thesis-FINAL.pdf.

S. Dieleman and B. Schrauwen. End-to-end learning for music audio. In *Proceedings of the 29th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6964–6968, May 2014. doi:10.1109/ICASSP.2014.6854950.

S. Dieleman, P. Braken, and B. Schrauwen. Audio-based music classification with a pretrained convolutional network. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, Miami, FL, USA, Oct. 2011. URL http://ismir2011.ismir.net/papers/PS6-3.pdf.

S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, D. M. de Almeida, B. McFee, H. Weideman, G. Takács, P. de Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu, G. French, and J. Degrave. Lasagne: First release., Aug. 2015. doi:10.5281/zenodo.27878. Project URL http://github.com/Lasagne/Lasagne.

S. Dixon. Onset detection revisited. In *Proceedings of the 9th International Conference on Digital Audio Effects (DAFx)*, pages 133–137, Montréal, Quebec, Canada, Sept. 2006. URL http://www.dafx.ca/proceedings/papers/p_133.pdf.

M. Dorfer, J. Schlüter, A. Vall, F. Korzeniowski, and G. Widmer. End-to-end cross-modality retrieval with CCA projections and pairwise ranking loss. *arXiv e-prints*, abs/1705.06979, May 2017. URL http://arxiv.org/abs/1705.06979.

J. S. Downie, A. F. Ehmann, M. Bay, and M. C. Jones. The Music Information Retrieval Evaluation eXchange: Some observations and insights. In Z. W. Raś and A. A. Wieczorkowska, editors, *Advances in Music Information Retrieval*, pages 93–115. Springer, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-11674-2_5.

S. Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962. ISSN 0022-247X. doi:http://dx.doi.org/10.1016/0022-247X(62)90004-5. URL http://www.sciencedirect.com/science/article/pii/0022247X62900045.

D. P. W. Ellis and G. E. Poliner. Identifying 'cover songs' with chroma features and dynamic programming beat tracking. In *Proceedings of the 32nd IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages 1429–1432, Honolulu, HI, USA, Apr. 2007. doi:10.1109/ICASSP.2007.367348. URL http://www.ee.columbia.edu/~dpwe/pubs/EllisP07-coversongs.pdf.

D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In D. van Dyk and M. Welling, editors, *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5 of *Proceedings of Machine Learning Research*, pages 153–160, Clearwater Beach, FL, USA, Apr. 2009. PMLR. URL http://proceedings.mlr.press/v5/erhan09a.html.

D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, Mar. 2010. ISSN 1532-4435. URL http://jmlr.org/papers/volume11/erhan10a/erhan10a.pdf.

F. Eyben, S. Böck, B. Schuller, and A. Graves. Universal onset detection with bidirectional long short-term memory neural networks. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 589–594, Utrecht, Netherlands, Aug. 2010. URL http://ismir2010.ismir.net/proceedings/ismir2010-101.pdf.

C. Faloutsos and K. I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 163–174, 1995. URL http://cs.baylor.edu/~lind/_mypaper/fastmap.ps.

G. T. Fechner. *Elemente der Psychophysik*, volume 1. Breitkopf und Härtel, Leipzig, Germany, 1860. URL http://archive.org/details/elementederpsych001fech.

A. Fischer and C. Igel. An introduction to restricted Boltzmann machines. In *Proceedings of the 17th Iberoamerican Congress on Pattern Recognition (CIARP)*, pages 14–36, Buenos Aires, Argentinia, 2012. URL http://image.diku.dk/igel/paper/AItRBM-proof.pdf.

D. FitzGerald. Harmonic/percussive separation using median filtering. In *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx)*, Graz, Austria, Sept. 2010. URL http://dafx10.iem.at/papers/DerryFitzGerald_DAFx10_P15.pdf.

H. Fletcher. Auditory patterns. *Reviews of Modern Physics*, 12:47–65, 1940.

H. Fletcher and W. A. Munson. Loudness, its definition, measurement and calculation. *The Journal of the Acoustical Society of America*, 5(2):82–108, 1933. doi:10.1121/1.1915637.

A. Flexer. A closer look on artist filters for musical genre classification. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 341–344, Vienna, Austria, 2007. URL http://ismir2007.ismir.net/proceedings/ISMIR2007_p341_flexer.pdf.

A. Flexer. On inter-rater agreement in audio music similarity. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, pages 245–250, Taipei, Taiwan, Oct. 2014. URL http://www.terasoft.com.tw/conf/ismir2014/proceedings/T045_256_Paper.pdf.

A. Flexer and T. Grill. The problem of limited inter-rater agreement in modelling music similarity. *Journal of New Music Research*, 45(3):239–251, July 2016. doi:10.1080/09298215.2016.1200631.

A. Flexer, D. Schnitzer, and J. Schlüter. A MIREX meta-analysis of hubness in audio music similarity. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR)*, Porto, Portugal, Oct. 2012. URL http://jan-schlueter.de/pubs/2012_ismir.pdf.

J. T. Foote. Content-based retrieval of music and audio. In J. C. C. Kuo, S. F. Chang, and V. N. Gudivada, editors, *Multimedia Storage and Archiving Systems II (Proceedings SPIE)*, volume 3229, pages 138–147, 1997. doi:10.1117/12.290336. URL http://rotorbrain.com/foote/papers/spie97.pdf.

J. T. Foote. Automatic audio segmentation using a measure of audio novelty. In *Proceedings of the 2000 IEEE International Conference on Multimedia and Expo (ICME)*, volume 1, pages 452–455, New York City, NY, USA, July 2000. doi:10.1109/ICME.2000.869637. URL http://rotorbrain.com/foote/papers/footeICME00.pdf.

J. T. Foote and M. L. Cooper. Media segmentation using self-similarity decomposition. In *Storage and Retrieval for Media Databases (Proceedings SPIE)*, volume 5021, pages 167–175, San Jose, CA, USA, Jan. 2003. doi:10.1117/12.476302. URL http://rotorbrain.com/foote/papers/SPIE02.pdf.

J. Foulds and E. Frank. A review of multi-instance learning assumptions. *Knowledge Engineering Review*, 25(1):1–25, Mar. 2010. doi:10.1017/S026988890999035X. URL http://www.cs.waikato.ac.nz/~ml/publications/2010/FouldsAndFrankMIreview.pdf.

T. Fujishima. Realtime chord recognition of musical sound: a system using common lisp music. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 464–467, 1999.

K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4): 193–202, 1980.

T. Garipov, D. Podoprikhin, A. Novikov, and D. P. Vetrov. Ultimate tensorization: compressing convolutional and FC layers alike. In *NIPS Workshop on Learning with Tensors: Why Now and How?*, Barcelona, Spain, Dec. 2016. URL http://arxiv.org/abs/1611.03214.

P. Ghahremani, V. Manohar, D. Povey, and S. Khudanpur. Acoustic modelling from the signal domain using CNNs. In *Proceedings of the 17th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 3434–3438, San Francisco, CA, USA, Sept. 2016. doi:10.21437/Interspeech.2016-1495. URL http://www.isca-speech.org/archive/Interspeech_2016/pdfs/1495.PDF.

A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 518–529, Edinburgh, United Kingdom, Sept. 1999. URL http://www.vldb.org/conf/1999/P49.pdf.

A. Giusti, D. C. Ciresan, J. Masci, L. M. Gambardella, and J. Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. Technical Report IDSIA-01-13, Dalle Molle Institute for Artificial Intelligence, Feb. 2013. URL http://arxiv.org/abs/1302.1700.

R. O. Gjerdingen and D. Perrott. Scanning the dial: The rapid recognition of music genres. *Journal of New Music Research*, 37(2):93–100, 2008. doi:10.1080/09298210802479268. URL http://faculty-web.at.northwestern.edu/music/gjerdingen/Papers/PubPapers/Scanning.pdf.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256,

Sardinia, Italy, May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, Apr. 2011. PMLR. URL http://proceedings.mlr.press/v15/glorot11a.html.

H. Goh, N. Thome, and M. Cord. Biasing restricted Boltzmann machines to manipulate latent selectivity and sparsity. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, Vancouver, Canada, Dec. 2010. URL http://webia.lip6.fr/~cord/pdfs/publis/NIPS2010_Goh.pdf.

Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proceedings of the 24th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 817–824, Colorado Springs, CO, USA, June 2011. doi:10.1109/CVPR.2011.5995432. URL http://www.cs.unc.edu/~lazebnik/publications/cvpr11_small_code.pdf.

I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio. Multi-prediction deep Boltzmann machines. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 548–556. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5024-multi-prediction-deep-boltzmann-machines.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. ISBN 9780262035613. URL http://www.deeplearningbook.org.

M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka. RWC music database: Popular, classical, and jazz music databases. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR)*, pages 287–288, Paris, France, Oct. 2002. URL http://ismir2002.ismir.net/proceedings/03-SP04-1.pdf.

F. Gouyon, S. Dixon, E. Pampalk, and G. Widmer. Evaluating rhythmic descriptors for musical genre classification. In *Proceedings of the 25th International Audio*

Engineering Society (AES) Conference, London, UK, June 2004. URL http://mtg.upf.edu/files/publications/AES25-GouyonDixonPampalkWidmer.pdf.

T. Grill and J. Schlüter. Music boundary detection using neural networks on spectrograms and self-similarity lag matrices. In *Proceedings of the 23rd European Signal Processing Conference (EUSIPCO)*, Nice, France, Aug. 2015a. URL http://jan-schlueter.de/pubs/2015_eusipco.pdf.

T. Grill and J. Schlüter. Music boundary detection using neural networks on combined features and two-level annotations. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, Málaga, Spain, Oct. 2015b. URL http://jan-schlueter.de/pubs/2015_ismir2.pdf.

T. Grill and J. Schlüter. Structural segmentation with convolutional neural networks MIREX submission. In *online Proceedings of the 11th Annual Music Information Retrieval Evaluation eXchange (MIREX)*, Málaga, Spain, Aug. 2015c. URL http://www.music-ir.org/mirex/abstracts/2015/GS1.pdf.

T. Grill and J. Schlüter. Two convolutional neural networks for bird detection in audio signals. In *Proceedings of the 25th European Signal Processing Conference (EUSIPCO)*, Kos Island, Greece, Aug. 2017. URL http://jan-schlueter.de/pubs/2017_eusipco.pdf. To appear.

S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pages 1737–1746, Lille, France, July 2015. PMLR. URL http://proceedings.mlr.press/v37/gupta15.html.

E. Gómez. *Tonal Description of Music Audio Signals*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Spain, 2006. URL http://mtg.upf.edu/system/files/publications/emilia-PhD-2006_0.pdf.

P. Hamel. *Apprentissage de représentations musicales à l'aide d'architectures profondes et multiéchelles*. PhD thesis, University of Montréal, Montréal, Canada, May 2012. doi:1866/8678.

P. Hamel and D. Eck. Learning features from music audio with deep belief networks. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 339–344, Utrecht, Netherlands, Aug. 2010. URL http://ismir2010.ismir.net/proceedings/ismir2010-58.pdf.

P. Hamel, S. Wood, and D. Eck. Automatic identification of instrument classes in polyphonic and poly-instrument audio. In *Proceedings of the 10th International*

Society for Music Information Retrieval Conference (ISMIR), pages 399–404, Kobe, Japan, Oct. 2009. URL http://ismir2009.ismir.net/proceedings/PS3-2.pdf.

P. Hamel, S. Lemieux, Y. Bengio, and D. Eck. Temporal pooling and multiscale learning for automatic annotation and ranking of music audio. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, Miami, FL, USA, Oct. 2011. URL http://ismir2011.ismir.net/papers/PS6-13.pdf.

P. Hamel, M. E. P. Davies, K. Yoshii, and M. Goto. Transfer learning in MIR: Sharing learned latent representations for music audio classification and similarity. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, pages 9–14, Curitiba, Brazil, Nov. 2013. URL http://www.ppgia.pucpr.br/ismir2013/wp-content/uploads/2013/09/76_Paper.pdf.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016. URL http://arxiv.org/abs/1510.00149.

J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 6–20, Berkeley, CA, USA, 1986. doi:10.1145/12130.12132. URL http://www.nada.kth.se/~johanh/largesmalldepth.pdf.

M. J. Hawley. *Structure out of sound*. PhD thesis, Massachusetts Institute of Technology, Cambridge, 1993. doi:1721.1/29068.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Washington, D.C., USA, Dec. 2015. doi:10.1109/ICCV.2015.123. URL http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf. Preprint http://arxiv.org/abs/1502.01852.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016a. doi:10.1109/CVPR.2016.90. URL http://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf. Preprint http://arxiv.org/abs/1512.03385.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Proceedings of the 14th European Conference on Computer Vision (ECCV)*, pages 630–645, Amsterdam, Netherlands, 2016b. Springer International Publishing. doi:10.1007/978-3-319-46493-0_38. Preprint http://arxiv.org/abs/1603.05027.

G. Heinzel, A. Rüdiger, and R. Schilling. Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows. Technical report, Max-Planck-Institut für Gravitationsphysik, Teilinstitut Hannover, 2002. URL http://holometer.fnal.gov/GH_FFT.pdf.

J. Heng, G. Cantarero, M. Elhilali, and C. J. Limb. Impaired perception of temporal fine structure and musical timbre in cochlear implant users. *Hearing Research*, 280(1–2):192–200, 2011. doi:10.1016/j.heares.2011.05.017.

L. Hertel, H. Phan, and A. Mertins. Comparing time and frequency domain for audio event recognition using deep learning. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 3407–3411, Vancouver, Canada, July 2016. doi:10.1109/IJCNN.2016.7727635. Preprint http://arxiv.org/abs/1603.05824.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002. doi:10.1162/089976602760128018. URL http://www.cs.utoronto.ca/~hinton/absps/nccd.pdf.

G. E. Hinton. To recognize shapes, first learn to generate images. In T. D. Paul Cisek and J. F. Kalaska, editors, *Computational Neuroscience: Theoretical Insights into Brain Function*, volume 165 of *Progress in Brain Research*, pages 535–547. Elsevier, 2007. doi:10.1016/S0079-6123(06)65034-6. URL http://www.cs.toronto.edu/~hinton/absps/montrealTR.pdf.

G. E. Hinton. A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 2010-003, Department of Computer Science, University of Toronto, 2010. URL http://www.csri.utoronto.ca/~hinton/absps/guideTR.pdf.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. URL http://www.cs.toronto.edu/~hinton/science.pdf.

G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006. ISSN 0899-7667.

doi:10.1162/neco.2006.18.7.1527. URL http://www.cs.toronto.edu/~hinton/absps/fastnc.pdf.

G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov. 2012a. doi:10.1109/MSP.2012.2205597. URL http://www.cs.toronto.edu/~asamir/papers/SPM_DNN_12.pdf.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv e-prints*, abs/1207.0580, July 2012b. URL http://arxiv.org/abs/1207.0580.

G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Workshop on Deep Learning and Representation Learning*, Montréal, Canada, Dec. 2014. URL http://arxiv.org/abs/1503.02531.

A. Holzapfel, Y. Stylianou, A. C. Gedik, and B. Bozkurt. Three dimensions of pitched instrument onset detection. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1517–1527, Aug. 2010. doi:10.1109/TASL.2009.2036298. URL http://www.ics.forth.gr/netlab/data/J12.pdf.

H. Homburg, I. Mierswa, B. Möller, K. Morik, and M. Wurst. A benchmark dataset for audio classification and clustering. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 528–531, London, UK, Sept. 2005. URL http://ismir2005.ismir.net/proceedings/2117.pdf.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989. doi:10.1016/0893-6080(89)90020-8. URL http://deeplearning.cs.cmu.edu/pdfs/Kornick_et_al.pdf.

L. Hou, D. Samaras, T. M. Kurç, Y. Gao, J. E. Davis, and J. H. Saltz. Efficient multiple instance convolutional neural networks for gigapixel resolution image classification. *arXiv e-prints*, abs/1504.07947v3, Apr. 2015. URL http://arxiv.org/abs/1504.07947v3.

K.-C. Hsu, C.-S. Lin, and T.-S. Chi. Sparse coding based music genre classification using spectro-temporal modulations. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, pages 744–750,

New York City, NY, USA, Aug. 2016. URL http://wp.nyu.edu/ismir2016/wp-content/uploads/sites/2294/2016/07/046_Paper.pdf.

G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, Honolulu, HI, USA, July 2017. URL http://openaccess.thecvf.com/content_cvpr_2017/papers/Huang_Densely_Connected_Convolutional_CVPR_2017_paper.pdf.

P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis. Singing-voice separation from monaural recordings using deep recurrent neural networks. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, pages 477–482, Taipei, Taiwan, Oct. 2014. URL http://www.terasoft.com.tw/conf/ismir2014/proceedings/T087_154_Paper.pdf.

E. J. Humphrey. *An Exploration of Deep Learning in Content-Based Music Informatics*. PhD thesis, New York University, New York City, NY, USA, 2015. URL http://github.com/ejhumphrey/dl4mir-dissertation.

E. J. Humphrey and J. P. Bello. Rethinking automatic chord recognition with convolutional neural networks. In *Proceedings of the 11th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 357–362, Boca Raton, FL, USA, Dec. 2012. doi:10.1109/ICMLA.2012.220.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, July 2015. PMLR. URL http://proceedings.mlr.press/v37/ioffe15.html.

B. Ionescu, I. Mironică, K. Seyerlehner, P. Knees, J. Schlüter, M. Schedl, H. Cucu, A. Buzo, and P. Lambert. ARF @ MediaEval 2012: multimodal video classification. In *MediaEval 2012 Workshop*, Pisa, Italy, Oct. 2012. URL http://jan-schlueter.de/pubs/2012_mediaeval2.pdf.

B. Ionescu, J. Schlüter, I. Mironică, and M. Schedl. A naïve mid-level concept-based fusion approach to violence detection in hollywood movies. In *Proceedings of the ACM International Conference on Multimedia Retrieval (ICMR)*, Dallas, TX, USA, Apr. 2013. URL http://jan-schlueter.de/pubs/2013_icmr.pdf.

T. Izumitani, R. Mukai, and K. Kashino. A background music detection method based on robust feature extraction. In *Proceedings of the 33rd IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 13–16, Las Vegas, NV, USA, Mar. 2008. doi:10.1109/ICASSP.2008.4517534.

N. Jaitly and G. E. Hinton. Vocal tract length perturbation (VTLP) improves speech recognition. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing (WDLASL)*, June 2013. URL http://www.cs.toronto.edu/~ndjaitly/jaitly-icml13.pdf.

J. H. Jensen, M. G. Christensen, and S. H. Jensen. A tempo-insensitive representation of rhythmic patterns. In *Proceedings of the 17th European Signal Processing Conference (EUSIPCO)*, pages 1509–1512, Glasgow, Scotland, UK, 2009. URL http://www.eurasip.org/Proceedings/Eusipco/Eusipco2009/contents/papers/1569189520.pdf.

N. Kanda, R. Takeda, and Y. Obuchi. Elastic spectral distortion for low resource speech recognition with deep neural networks. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 309–314, Olomouc, Czech Republic, Dec. 2013. doi:10.1109/ASRU.2013.6707748.

A. Karpathy, F.-F. Li, and J. Johnson. CS231n: Convolutional neural networks for visual recognition, 2016. URL http://cs231n.github.io/.

J. D. Keeler, D. E. Rumelhart, and W. K. Leow. Integrated segmentation and recognition of hand-printed numerals. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 557–563. Morgan-Kaufmann, 1991. URL http://papers.nips.cc/paper/397-integrated-segmentation-and-recognition-of-hand-printed-numerals.

N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, Toulon, France, Apr. 2017. URL http://arxiv.org/abs/1609.04836.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015. URL http://arxiv.org/abs/1412.6980.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, Banff, Canada, Apr. 2014. URL http://arxiv.org/abs/1312.6114.

G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *arXiv e-prints*, abs/1706.02515, June 2017. URL http://arxiv.org/abs/1706.02515.

A. P. Klapuri, A. J. Eronen, and J. T. Astola. Analysis of the meter of acoustic musical signals. *IEEE Transactions on Audio, Speech, and Language Processing*,

14(1):342–355, Jan. 2006. doi:10.1109/TSA.2005.854090. URL http://www.cs.tut.fi/sgn/arg/klap/sapmeter.pdf.

W. Koenig. A new frequency scale for acoustic measurements. *Bell Telephone Laboratory Record*, 27:299–301, 1949.

P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent initializations of convolutional neural networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016. URL http://arxiv.org/abs/1511.06856.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009. URL http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

A. Krizhevsky and G. E. Hinton. Using very deep autoencoders for content-based image retrieval. In *Proceedings of the 19th European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, Apr. 2011. URL http://www.cs.toronto.edu/~hinton/absps/esann-deep-final.pdf.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

A. Lacoste and D. Eck. A supervised classification algorithm for note onset detection. *EURASIP Journal on Advances in Signal Processing*, 2007(1):043745, 2006. doi:10.1155/2007/43745. URL http://link.springer.com/content/pdf/10.1155/2007/43745.pdf.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov. 1998a. URL http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. Efficient BackProp. In G. B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, Berlin, Heidelberg, 1998b. doi:10.1007/3-540-49430-8_2. URL http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf.

Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 17th IEEE Computer Society Conference on Computer Vision and Pattern*

Recognition (CVPR), volume 2, pages 97–104, Washington, D.C., USA, June 2004. doi:10.1109/CVPR.2004.1315150. URL http://yann.lecun.com/exdb/publis/pdf/lecun-04.pdf.

D.-H. Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *ICML Workshop on Challenges in Representation Learning*, Atlanta, GA, USA, June 2013. URL http://deeplearning.net/wp-content/uploads/2013/03/pseudo_label_final.pdf.

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 609–616, Montréal, Quebec, Canada, 2009a. doi:10.1145/1553374.1553453. URL http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf.

H. Lee, P. Pham, Y. Largman, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1096–1104. Curran Associates, Inc., 2009b. URL http://papers.nips.cc/paper/3674-unsupervised-feature-learning-for-audio-classification-using-convolutional-deep-belief-networks.

S. Leglaive, R. Hennequin, and R. Badeau. Singing voice detection with deep recurrent neural networks. In *Proceedings of the 40th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 121–125, Brisbane, Australia, Apr. 2015. doi:10.1109/ICASSP.2015.7177944. URL http://hal.archives-ouvertes.fr/hal-01110035.

B. Lehner, G. Widmer, and R. Sonnleitner. On the reduction of false positives in singing voice detection. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 7530–7534, Florence, Italy, May 2014. doi:10.1109/ICASSP.2014.6855054. URL http://www.cp.jku.at/research/papers/Lehner_etal_ICASSP_2014.pdf.

B. Lehner, G. Widmer, and S. Böck. A low-latency, real-time-capable singing voice detection method with LSTM recurrent neural networks. In *Proceedings of the 23rd European Signal Processing Conference (EUSIPCO)*, pages 21–25, Nice, France, Aug. 2015. doi:10.1109/EUSIPCO.2015.7362337. URL http://www.eurasip.org/Proceedings/Eusipco/Eusipco2015/papers/1570097385.pdf.

M. Levy and M. Sandler. Structural segmentation of musical audio by constrained clustering. *IEEE Transactions on Audio, Speech, and Language Processing*, 16 (2):318–326, Feb. 2008. doi:10.1109/TASL.2007.910781.

T. L. Li and A. B. Chan. Genre classification and the invariance of MFCC features to key and tempo. In *Proceedings of the 17th International Conference on Multimedia Modeling (MMM)*, pages 317–327, Taipei, Taiwan, Jan. 2011. doi:10.1007/978-3-642-17832-0_30. URL http://visal.cs.cityu.edu.hk/static/pubs/conf/mmm11-mfcc.pdf.

T. L. Li, A. B. Chan, and A. H. Chun. Automatic musical pattern feature extraction using convolutional neural network. In *International MultiConference of Engineers and Computer Scientists (IMECS)*, Hong Kong, Mar. 2010. URL http://www.iaeng.org/publication/IMECS2010/IMECS2010_pp546-550.pdf.

X.-B. Li, F. K. Soong, T. A. Myrvoll, and R.-H. Wang. Optimal clustering and non-uniform allocation of gaussian kernels in scalar dimension for HMM compression. In *Proceedings of the 30th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 669–672, Philadelphia, PA, USA, Mar. 2005. doi:10.1109/ICASSP.2005.1415202.

M. Lin, Q. Chen, and S. Yan. Network in network. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, Banff, Canada, Apr. 2014. URL http://arxiv.org/abs/1312.4400.

S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki, 1970.

C. Liu, L. Xie, and H. Meng. Classification of music and speech in mandarin news broadcasts. In *Proceedings of the 9th National Conference on Man-Machine Speech Communication (NCMMSC)*, Huangshan, Anhui, China, 2007. URL http://liuchuan.org/pub/NCMMSC07.pdf.

J.-Y. Liu and Y.-H. Yang. Event localization in music auto-tagging. In *Proceedings of the 24th ACM International Conference on Multimedia (ACMMM)*, pages 1048–1057, Amsterdam, Netherlands, Oct. 2016. doi:10.1145/2964284.2964292. URL http://mac.citi.sinica.edu.tw/~yang/pub/liu16mm.pdf.

T. Liu, A. W. Moore, K. Yang, and A. G. Gray. An investigation of practical approximate nearest neighbor algorithms. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 825–832. MIT Press, 2005. URL http://papers.nips.cc/paper/2666-an-investigation-of-practical-approximate-nearest-neighbor-algorithms.

A. Liutkus, D. Fitzgerald, Z. Rafii, B. Pardo, and L. Daudet. Kernel additive models for source separation. *IEEE Transactions on Signal Processing*, 62(16):4298–4310, Aug. 2014. doi:10.1109/TSP.2014.2332434. URL http://hal.archives-ouvertes.fr/hal-01011044.

A. Liutkus, D. Fitzgerald, and Z. Rafii. Scalable audio separation with light kernel additive modelling. In *Proceedings of the 40th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 76–80, Brisbane, Australia, Apr. 2015. doi:10.1109/ICASSP.2015.7177935. URL http://hal.archives-ouvertes.fr/hal-01114890.

B. Logan. Mel frequency cepstral coefficients for music modeling. In *Proceedings of the 1st International Symposium on Music Information Retrieval (ISMIR)*, Plymouth, MA, USA, 2000.

B. Logan and S. Chu. Music summarization using key phrases. In *Proceedings of the 25th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 749–752, Istanbul, Turkey, June 2000.

C. Lorenz. Untersuchungen über die Auffassung von Tondistanzen. *Wundts Philosophische Studien*, 6(1):26–103, 1890. URL http://echo.mpiwg-berlin.mpg.de/MPIWG:RB5H7KM1.

V. Lostanlen and C.-E. Cella. Deep convolutional networks on the pitch spiral for music instrument recognition. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, pages 612–618, New York City, United States, Aug. 2016. URL http://wp.nyu.edu/ismir2016/wp-content/uploads/sites/2294/2016/07/093_Paper.pdf. Preprint http://arxiv.org/abs/1605.06644.

L. Lu, M. Wang, and H.-J. Zhang. Repeating pattern discovery and structure analysis from acoustic music data. In *Proceedings of the 6th ACM SIGMM International Workshop on Multimedia Information Retrieval*, pages 275–282, New York City, NY, USA, Oct. 2004. ACM. doi:10.1145/1026711.1026756. URL http://www.microsoft.com/en-us/research/publication/repeating-pattern-discovery-and-structure-analysis-from-acoustic-music-data/.

Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multiprobe LSH: Efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 950–961, Vienna, Austria, Sept. 2007. URL http://vldb.org/conf/2007/papers/research/p950-lv.pdf.

R. G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall, 3rd edition, 2010. ISBN 0137027419.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 315–323, Atlanta, GA, USA, June 2013. URL http://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

J. Makhoul and L. Cosell. LPCW: An LPC vocoder with linear predictive spectral warping. In *Proceedings of the 1st IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 466–469, Philadelphia, PA, USA, Apr. 1976.

M. I. Mandel and D. P. W. Ellis. Song-level features and support vector machines for music classification. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 594–599, London, United Kingdom, Sept. 2005. URL http://ismir2005.ismir.net/proceedings/1106.pdf.

M. I. Mandel and D. P. W. Ellis. Multiple-instance learning for music information retrieval. In *Proceedings of the 9th International Society for Music Information Retrieval Conference (ISMIR)*, pages 577–582, Philadelphia, PA, USA, Sept. 2008. URL http://ismir2008.ismir.net/papers/ISMIR2008_205.pdf.

L. E. Marks and M. Florentine. Measurement of loudness, part I: Methods, problems, and pitfalls. In M. Florentine, A. N. Popper, and R. R. Fay, editors, *Loudness*, pages 17–56. Springer New York, New York, NY, 2011. ISBN 978-1-4419-6712-1. doi:10.1007/978-1-4419-6712-1_2.

M. Marolt, A. Kavcic, M. Privosnik, and S. Divjak. On detecting note onsets in piano music. In *Proceedings of the 11th IEEE Mediterranean Electrotechnical Conference (MELECON)*, pages 385–389, Cairo, Egypt, May 2002. doi:10.1109/MELECON.2002.1014600.

J. Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 735–742, Haifa, Israel, June 2010. URL http://icml2010.haifa.il.ibm.com/papers/458.pdf.

B. Mathieu, S. Essid, T. Fillon, J. Prado, and G. Richard. Yaafe, an easy to use and efficient audio feature extraction software. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 441–445, Utrecht, Netherlands, Aug. 2010. URL http://ismir2010.ismir.net/proceedings/ismir2010-75.pdf. Project URL http://github.com/Yaafe/Yaafe.

W. J. Matthews and W. H. Meck. Time perception: the bad news and the good. *Wiley Interdisciplinary Reviews. Cognitive Science*, 5(4):429–446, 2014. doi:10.1002/wcs.1298. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4142010.

M. Mauch, K. C. Noland, and S. Dixon. Using musical structure to enhance automatic chord transcription. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, pages 231–236, Kobe, Japan, Oct. 2009. URL http://ismir2009.ismir.net/proceedings/PS2-7.pdf.

M. Mauch, H. Fujihara, K. Yoshii, and M. Goto. Timbre and melody features for the recognition of vocal activity and instrumental solos in polyphonic music. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 233–238, Miami, FL, USA, Oct. 2011. URL http://ismir2011.ismir.net/papers/PS2-11.pdf.

B. McFee and D. P. W. Ellis. Learning to segment songs with ordinal linear discriminant analysis. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5197–5201, Florence, Italy, May 2014. doi:10.1109/ICASSP.2014.6854594. URL http://www.ee.columbia.edu/~dpwe/pubs/McFeeE14-segments.pdf.

B. McFee and G. Lanckriet. Large-scale music similarity search with spatial trees. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 55–60, Miami, FL, USA, Oct. 2011. URL http://ismir2011.ismir.net/papers/PS1-3.pdf.

B. McFee, E. J. Humphrey, and J. P. Bello. A software framework for musical data augmentation. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, pages 248–254, Màlaga, Spain, Oct. 2015. URL http://ismir2015.uma.es/articles/228_Paper.pdf.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Computation*, 22(6):1473–1492, 2010. doi:10.1162/neco.2010.01-09-953. URL http://www.iro.umontreal.ca/~memisevr/pubs/UTML-TR-2009-003.pdf.

N. Mesgarani, M. Slaney, and S. Shamma. Discrimination of speech from nonspeech based on multiscale spectro-temporal modulations. *IEEE Transactions on Audio, Speech and Language Processing*, 14(3):920–930, May 2006. doi:10.1109/TSA.2005.858055. URL http://slaney.org/malcolm/yahoo/Mesgarani2006-MultiscaleModulationsTASLP.pdf.

K. Minami, A. Akutsu, H. Hamada, and Y. Tonomura. Video handling with music and speech detection. *IEEE Multimedia*, 5(3):17–25, July 1998. doi:10.1109/93.713301.

D. Mishkin and J. Matas. All you need is a good init. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016. URL http://arxiv.org/abs/1511.06422.

D. Mishkin, N. Sergievskiy, and J. Matas. Systematic evaluation of CNN advances on the ImageNet. *arXiv e-prints*, abs/1606.02228, 2016. URL http://arxiv.org/abs/1606.02228.

V. Mnih. CUDAMat: a CUDA-based matrix class for python. Technical Report UTML TR 2009–004, Department of Computer Science, University of Toronto, Nov. 2009. URL http://www.cs.toronto.edu/~vmnih/docs/cudamat_tr.pdf. Project URL http://github.com/cudamat/cudamat.

A.-r. Mohamed, G. E. Dahl, and G. E. Hinton. Deep belief networks for phone recognition. In *NIPS Workshop on Deep Learning for Speech Recognition*, 2009. URL http://www.cs.toronto.edu/~asamir/papers/NIPS09.pdf.

A.-r. Mohamed, G. E. Dahl, and G. E. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20 (1):14–22, Jan. 2012. doi:10.1109/TASL.2011.2109382. URL http://www.cs.toronto.edu/~asamir/papers/speechDBN_jrnl.pdf.

S. Molau, M. Pitz, R. Schlüter, and H. Ney. Computing mel-frequency cepstral coefficients on the power spectrum. In *Proceedings of the 26th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 73–76, Salt Lake City, UT, USA, May 2001. doi:10.1109/ICASSP.2001.940770. URL http://www-i6.informatik.rwth-aachen.de/publications/download/474/Molau-ICASSP-2001.pdf.

G. Montavon and K.-R. Müller. Deep Boltzmann machines and the centering trick. In G. Montavon, G. B. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 621–637. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-35288-1. doi:10.1007/978-3-642-35289-8_33.

M. Müller. *Fundamentals of Music Processing – Audio, Analysis, Algorithms, Applications*. Springer Verlag, 2015. ISBN 3319219448.

J. Nam, J. Ngiam, H. Lee, and M. Slaney. A classification-based polyphonic piano transcription approach using learned feature representations. In *Proceedings*

of the 12th International Society for Music Information Retrieval Conference (ISMIR), Miami, FL, USA, Oct. 2011. URL http://ismir2011.ismir.net/papers/PS2-1.pdf.

J. Nam, J. Herrera, M. Slaney, and J. Smith. Learning sparse feature representations for music annotation and retrieval. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR)*, pages 565–570, Porto, Portugal, Oct. 2012. URL http://ismir2012.ismir.net/event/papers/565_ISMIR_2012.pdf.

Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

J. Ngiam, Z. Chen, P. W. Koh, and A. Y. Ng. Learning deep energy models. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 1105–1112, Bellevue, WA, USA, June 2011. URL http://ai.stanford.edu/~ang/papers/icml11-DeepEnergyModels.pdf.

M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL http://neuralnetworksanddeeplearning.com.

M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov. Hamming distance metric learning. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1061–1069. Curran Associates, Inc., 2012a. URL http://papers.nips.cc/paper/4808-hamming-distance-metric-learning.

M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *Proceedings of the 25th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3108–3115, 2012b. URL http://www.cs.toronto.edu/~norouzi/research/papers/multi_index_hashing.pdf.

A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 442–450. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5787-tensorizing-neural-networks.

A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Pearson, 3rd edition, 2009. ISBN 0131988425.

C. Osendorfer, J. Schlüter, J. Schmidhuber, and P. van der Smagt. Unsupervised learning of low-level audio features for music similarity estimation. In *ICML Workshop on Learning Architectures, Representations, and Optimization*

for Speech and Visual Information Processing, Bellevue, WA, USA, June 2011. URL http://jan-schlueter.de/pubs/2011_icmlws.pdf.

E. Pampalk. *Computational Models of Music Similarity and their Application in Music Information Retrieval*. PhD thesis, Vienna University of Technology, Vienna, Austria, Mar. 2006. URL http://www.ofai.at/~elias.pampalk/publications/pampalk06thesis.pdf.

E. Pampalk, A. Rauber, and D. Merkl. Content-based organization and visualization of music archives. In *Proceedings of the 10th ACM International Conference on Multimedia*, pages 570–579, Juan-les-Pins, France, 2002. doi:10.1145/641007.641121. URL http://www.ofai.at/~elias.pampalk/publications/pam_mm02.pdf.

H. Papadopoulos and G. Peeters. Joint estimation of chords and downbeats from an audio signal. *IEEE Transactions on Audio, Speech, and Language Processing*, 19 (1):138–152, Jan. 2011. ISSN 1558-7916. doi:10.1109/TASL.2010.2045236. URL http://hal.archives-ouvertes.fr/IRCAM/hal-00525172.

J. Paulus and A. Klapuri. Music structure analysis by finding repeated parts. In *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia (AMCMM)*, pages 59–68, Santa Barbara, CA, USA, 2006. ACM. doi:10.1145/1178723.1178733. URL http://paulus.kapsi.fi/pubs/amcmm718-paulus_copy.pdf.

J. Paulus and A. Klapuri. Acoustic features for music piece structure analysis. In *Proceedings of the 11th International Conference on Digital Audio Effects (DAFx)*, pages 309–312, Espoo, Finland, Sept. 2008. URL http://paulus.kapsi.fi/pubs/dafx08_paulus.pdf.

J. Paulus and A. Klapuri. Music structure analysis using a probabilistic fitness measure and a greedy search algorithm. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(6):1159–1170, Aug. 2009. doi:10.1109/TASL.2009.2020533. URL http://paulus.kapsi.fi/pubs/paulus-taslp09-copy.pdf.

J. Paulus, M. Müller, and A. Klapuri. Audio-based music structure analysis. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 625–636, Utrecht, Netherlands, Aug. 2010. URL http://ismir2010.ismir.net/proceedings/ismir2010-107.pdf.

G. Peeters. A large set of audio features for sound description (similarity and classification) in the cuidado project. Technical report, IRCAM,

Paris, France, 2004. URL http://recherche.ircam.fr/equipes/analyse-synthese/peeters/ARTICLES/Peeters_2003_cuidadoaudiofeatures.pdf.

S. Pfeiffer. Pause concepts for audio segmentation at different semantic levels. In *Proceedings of the 9th ACM International Conference on Multimedia (ACMMM)*, pages 187–193, Ottawa, Canada, Sept. 2001. doi:10.1145/500141.500171.

P. O. Pinheiro and R. Collobert. From image-level to pixel-level labeling with convolutional networks. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1713–1721, Boston, MA, USA, June 2015. doi:10.1109/CVPR.2015.7298780. URL http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Pinheiro_From_Image-Level_to_2015_CVPR_paper.pdf.

J. Pinquier, C. Sénac, and R. André-Obrecht. Speech and music classification in audio documents. In *Proceedings of the 27th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages 4164–4167, Orlando, FL, USA, May 2002. doi:10.1109/ICASSP.2002.5745593.

T. Pohle. *Automatic Characterization of Music for Intuitive Retrieval*. PhD thesis, Johannes Kepler University, Linz, Austria, Jan. 2010. URL http://www.cp.jku.at/research/papers/Pohle_Dissertation_2009.pdf.

T. Pohle, D. Schnitzer, M. Schedl, P. Knees, and G. Widmer. On rhythm and general music similarity. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, pages 525–530, Kobe, Japan, 2009. URL http://ismir2009.ismir.net/proceedings/OS6-1.pdf.

L. C. W. Pols. *Spectral analysis and identification of dutch vowels in monosyllabic words*. PhD thesis, Free University, Amsterdam, Netherlands, 1977.

S. M. Prentiss, D. R. Friedland, T. Fullmer, A. Crane, T. Stoddard, and C. L. Runge. Temporal and spectral contributions to musical instrument identification and discrimination among cochlear implant users. *World Journal of Otorhinolaryngology - Head and Neck Surgery*, 2(3):148–156, 2016. doi:10.1016/j.wjorl.2016.09.001. Special Issue: Otology and Neurotology.

W. Preyer. *Über die Grenzen der Tonwahrnehmung*. Verlag von Hermann Dufft, Jena, Germany, 1876. URL http://archive.org/details/berdiegrenzende00preygoog.

S. Qu, J. Li, W. Dai, and S. Das. Learning filter banks using deep learning for acoustic signals. *arXiv e-prints*, abs/1611.09526, Nov. 2016. URL http://arxiv.org/abs/1611.09526.

D. Rafailidis, A. Nanopoulos, and Y. Manolopoulos. Nonlinear dimensionality reduction for efficient and effective audio similarity searching. *Multimedia Tools and Applications*, 51(3):881–895, 2011. doi:10.1007/s11042-009-0420-7. URL http://delab.csd.auth.gr/~draf/papers/NLDR.pdf.

C. Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, Columbia University, New York City, NY, USA, July 2016. URL http://colinraffel.com/publications/thesis.pdf.

Z. Rafii and B. Pardo. REpeating Pattern Extraction Technique (REPET): A simple method for music/voice separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(1):73–84, Jan. 2013. doi:10.1109/TASL.2012.2213249. URL http://www.zafarrafii.com/doc/Rafii-Pardo - REpeating Pattern Extraction Technique (REPET) A Simple Method for Music-Voice Separation - TALSP 2013.pdf.

A. Ragni, K. M. Knill, S. P. Rath, and M. J. F. Gales. Data augmentation for low resource languages. In H. Li, H. M. Meng, B. Ma, E. Chng, and L. Xie, editors, *Proceedings of the 15th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 810–814, Singapore, Sept. 2014. ISCA. URL http://www.isca-speech.org/archive/interspeech_2014/i14_0810.html.

M. Ramona, G. Richard, and B. David. Vocal detection in music with support vector machines. In *Proceedings of the 33rd IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1885–1888, Las Vegas, NV, USA, Mar. 2008. doi:10.1109/ICASSP.2008.4518002. URL http://www.mathieuramona.com/uploads/Main/bibA03.pdf.

M. Ranzato and G. E. Hinton. Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *Proceedings of the 23rd IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2551–2558, June 2010. doi:10.1109/CVPR.2010.5539962. URL http://www.csri.utoronto.ca/~hinton/absps/ranzato_cvpr2010.pdf.

M. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1185–1192. Curran Associates, Inc., 2008. URL http://papers.nips.cc/paper/3363-sparse-feature-learning-for-deep-belief-networks.

M. Ranzato, A. Krizhevsky, and G. E. Hinton. Factored 3-way restricted Boltzmann machines for modeling natural images. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Fort Lauderdale, FL, USA, Apr. 2010. URL http://www.cs.toronto.edu/~hinton/absps/ranzato_aistats2010.pdf.

A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3546–3554. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5947-semi-supervised-learning-with-ladder-networks.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-net: ImageNet classification using binary convolutional neural networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Proceedings of the 14th European Conference on Computer Vision (ECCV)*, pages 525–542, Amsterdam, Netherlands, 2016. Springer International Publishing. ISBN 978-3-319-46493-0. doi:10.1007/978-3-319-46493-0_32. URL http://pjreddie.com/media/files/papers/xnor.pdf. Preprint http://arxiv.org/abs/1603.05279.

L. F. Richardson and J. S. Ross. Loudness and telephone current. *The Journal of General Psychology*, 3(2):288–306, 1930. doi:10.1080/00221309.1930.9918206.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 2(65):386–408, Nov. 1958. URL http://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. URL http://www.cs.utoronto.ca/~hinton/absps/pdp8.pdf.

H. B. Sailor and H. A. Patil. Novel unsupervised auditory filterbank learning using convolutional RBM for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(12):2341–2353, Dec. 2016. doi:10.1109/TASLP.2016.2607341.

T. N. Sainath, B. Kingsbury, A.-r. Mohamed, and B. Ramabhadran. Learning filter banks within a deep neural network framework. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 297–302, Olomouc, Czech

Republic, Dec. 2013a. doi:10.1109/ASRU.2013.6707746. URL http://sites.google.com/site/tsainath/tsainath_filterLearning_asru2013.pdf.

T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6655–6659, Vancouver, Canada, May 2013b. doi:10.1109/ICASSP.2013.6638949. URL http://sites.google.com/site/tsainath/tsainath_lowRank.pdf.

T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran. Deep convolutional neural networks for LVCSR. In *Proceedings of the 38th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 8614–8618, Vancouver, Canada, May 2013c. doi:10.1109/ICASSP.2013.6639347. URL http://www.cs.toronto.edu/~asamir/papers/icassp13_cnn.pdf.

T. N. Sainath, R. J. Weiss, A. W. Senior, K. W. Wilson, and O. Vinyals. Learning the speech front-end with raw waveform CLDNNs. In *Proceedings of the 16th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Dresden, Germany, Sept. 2015. URL http://www.isca-speech.org/archive/interspeech_2015/papers/i15_0001.pdf.

R. R. Salakhutdinov. Learning deep Boltzmann machines using adaptive MCMC. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 943–950, Haifa, Israel, June 2010. URL http://www.cs.cmu.edu/~rsalakhu/papers/adapt.pdf.

R. R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In D. van Dyk and M. Welling, editors, *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5 of *Proceedings of Machine Learning Research*, pages 448–455, Clearwater Beach, FL, USA, Apr. 2009a. PMLR. URL http://proceedings.mlr.press/v5/salakhutdinov09a.html.

R. R. Salakhutdinov and G. E. Hinton. Semantic hashing. *International Journal of Approximative Reasoning*, 50(7):969–978, July 2009b. doi:10.1016/j.ijar.2008.11.006. URL http://www.cs.utoronto.ca/~rsalakhu/papers/semantic_final.pdf.

R. R. Salakhutdinov and G. E. Hinton. A better way to pretrain deep Boltzmann machines. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2447–2455. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4610-a-better-way-to-pretrain-deep-boltzmann-machines.

T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 901–901. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6114-weight-normalization-a-simple-reparameterization-to-accelerate-training-of-deep-neural-networks.

J. Saunders. Real-time discrimination of broadcast speech/music. In *Proceedings of the 21st IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 993–996, Atlanta, GA, USA, May 1996. doi:10.1109/ICASSP.1996.543290. URL http://www.ee.columbia.edu/~dpwe/papers/Saun96-spmus.pdf.

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, Banff, Canada, Apr. 2014. URL http://arxiv.org/abs/1312.6120.

E. Scheirer and M. Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. In *Proceedings of the 22nd IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1331–1334, Munich, Germany, Apr. 1997. doi:10.1109/ICASSP.1997.596192. URL http://www.ee.columbia.edu/~dpwe/papers/ScheiS97-mussp.pdf.

J. Schlüter. Unsupervised audio feature extraction for music similarity estimation. Master's thesis, Technische Universität München, Munich, Germany, Oct. 2011. URL http://jan-schlueter.de/pubs/msc.pdf.

J. Schlüter. Learning binary codes for efficient large-scale music similarity search. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, Curitiba, Brazil, Nov. 2013. URL http://jan-schlueter.de/pubs/2013_ismir.pdf.

J. Schlüter. Restricted Boltzmann machine derivations. Technical Report TR-2014-13, Österreichisches Forschungsinstitut für Artificial Intelligence (OFAI), Vienna, Austria, Mar. 2014. URL http://jan-schlueter.de/pubs/2014_techrep_rbm.pdf.

J. Schlüter. Learning to pinpoint singing voice from weakly labeled examples. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, New York City, NY, USA, Aug. 2016. URL http://jan-schlueter.de/pubs/2016_ismir.pdf.

J. Schlüter and S. Böck. Musical onset detection with convolutional neural networks. In *6th International Workshop on Machine Learning and Music (MML), in conjunction with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, Prague, Czech Republic, Sept. 2013. URL http://jan-schlueter.de/pubs/2013_mml.pdf.

J. Schlüter and S. Böck. Improved musical onset detection with convolutional neural networks. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6979–6983, Florence, Italy, May 2014. doi:10.1109/ICASSP.2014.6854953. URL http://jan-schlueter.de/pubs/2014_icassp.pdf.

J. Schlüter and T. Grill. Exploring data augmentation for improved singing voice detection with neural networks. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, Málaga, Spain, Oct. 2015. URL http://jan-schlueter.de/pubs/2015_ismir.pdf.

J. Schlüter and R. Sonnleitner. Unsupervised feature learning for speech and music detection in radio broadcasts. In *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx)*, York, UK, Sept. 2012. URL http://jan-schlueter.de/pubs/2012_dafx.pdf.

J. Schlüter, L. Cabac, and D. Moldt. Adding runtime net manipulation features to MulanViewer. In *15th German Workshop on Algorithms and Tools for Petri Nets (AWPN)*, Rostock, Germany, Sept. 2008. URL http://ceur-ws.org/Vol-380/paper14.pdf.

J. Schlüter, B. Ionescu, I. Mironică, and M. Schedl. ARF @ MediaEval 2012: an uninformed approach to violence detection in hollywood movies. In *MediaEval 2012 Workshop*, Pisa, Italy, Oct. 2012. URL http://jan-schlueter.de/pubs/2012_mediaeval.pdf.

J. Schlüter and C. Osendorfer. Music similarity estimation with the mean-covariance restricted Boltzmann machine. In *Proceedings of the 10th International Conference on Machine Learning and Applications (ICMLA)*, Honolulu, HI, USA, Dec. 2011. URL http://jan-schlueter.de/pubs/2011_icmla.pdf.

E. M. Schmidt. *Modeling and Predicting Emotion in Music*. PhD thesis, Drexel University, Philadelphia, PA, USA, Sept. 2012. doi:1860/3997.

D. Schnitzer. Mirage – high-performance music similarity computation and automatic playlist generation. Master's thesis, Vienna University of Technology, 2007. URL http://www.schnitzer.at/dominik/pdf/mirage.pdf.

D. Schnitzer. *Indexing Content-Based Music Similarity Models for Fast Retrieval in Massive Databases*. PhD thesis, Johannes Kepler Univ. Linz, Austria, Feb. 2011. URL http://www.schnitzer.at/dominik/pdf/dominikschnitzer2011thesis.pdf.

D. Schnitzer, A. Flexer, and G. Widmer. A filter-and-refine indexing method for fast similarity search in millions of music tracks. In *Proceedings of the 10th International Society of Music Information Retrieval Conference (ISMIR)*, Kobe, Japan, Oct. 2009. URL http://ismir2009.ismir.net/proceedings/OS6-3.pdf.

D. Schnitzer, A. Flexer, and J. Schlüter. The relation of hubs to the Doddington zoo in speaker verification. In *Proceedings of the 21st European Signal Processing Conference (EUSIPCO)*, Marrakech, Morocco, Sept. 2013. URL http://jan-schlueter.de/pubs/2013_eusipco.pdf.

H. Seki, K. Yamamoto, and S. Nakagawa. A deep neural network integrated with filterbank learning for speech recognition. In *Proceedings of the 42nd IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5480–5484, New Orleans, LA, USA, Mar. 2017. doi:10.1109/ICASSP.2017.7953204. URL http://www.slp.ics.tut.ac.jp/nakagawa/pdfs/seki.icassp.2017.pdf.

T. Sercu and V. Goel. Dense prediction on sequences with time-dilated convolutions for speech recognition. In *NIPS Workshop on End-to-end Learning for Speech and Audio Processing*, Barcelona, Spain, Nov. 2016. URL http://arxiv.org/abs/1611.09288.

T. Sercu, C. Puhrsch, B. Kingsbury, and Y. LeCun. Very deep multilingual convolutional neural networks for LVCSR. In *Proceedings of the 41st IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4955–4959, Mar. 2016. doi:10.1109/ICASSP.2016.7472620. Preprint http://arxiv.org/abs/1509.08967.

P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, Banff, Canada, Apr. 2014. URL http://arxiv.org/abs/1312.6229.

J. Serra, M. Müller, P. Grosche, and J. L. Arcos. Unsupervised detection of music boundaries by time series structure features. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 1613–1619. Association for the Advancement of Artificial Intelligence, 2012. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/download/4907/5309.

K. Seyerlehner. *Content-Based Music Recommender Systems: Beyond simple Frame-Level Audio Similarity*. PhD thesis, Johannes Kepler University, Linz, Austria, Dec. 2010. URL http://www.cp.jku.at/people/seyerlehner/supervised/seyerlehner_phd.pdf.

K. Seyerlehner, T. Pohle, M. Schedl, and G. Widmer. Automatic music detection in television productions. In *Proceedings of the 10th International Conference on Digital Audio Effects (DAFx)*, Bordeaux, France, 2007. URL http://www.cp.jku.at/research/papers/Seyerlehner_etal_DAFx_2007.pdf.

K. Seyerlehner, M. Schedl, T. Pohle, and P. Knees. Using block-level features for genre classification, tag classification and music similarity estimation. In *online Proceedings of the 6th Annual Music Information Retrieval Evaluation eXchange (MIREX)*, Utrecht, Netherlands, Aug. 2010a. URL http://www.music-ir.org/mirex/abstracts/2010/SSPK1.pdf.

K. Seyerlehner, G. Widmer, and T. Pohle. Fusing block-level features for music similarity estimation. In *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx)*, Graz, Austria, Sept. 2010b. URL http://dafx10.iem.at/proceedings/papers/SeyerlehnerWidmerPohle_DAFx10_P31.pdf.

K. Seyerlehner, G. Widmer, M. Schedl, and P. Knees. Automatic music tag classification based on block-level features. In *Proceedings of the 7th Sound and Music Computing Conference (SMC)*, Barcelona, Spain, July 2010c. URL http://smcnetwork.org/files/proceedings/2010/19.pdf.

S. Sigtia. *Neural Networks for Analysing Music and Environmental Audio*. PhD thesis, Queen Mary University of London, London, United Kingdom, Nov. 2016. URL http://www.eecs.qmul.ac.uk/~simond/phd/SiddharthSigtia-PhD-Thesis.pdf.

S. Sigtia and S. Dixon. Improved music feature learning with deep neural networks. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6959–6963, Florence, Italy, May 2014. doi:10.1109/ICASSP.2014.6854949. URL http://www.eecs.qmul.ac.uk/~sss31/Pubs/ICASSP-2014.pdf.

S. Sigtia, E. Benetos, N. Boulanger-Lewandowski, T. Weyde, A. S. d'Avila Garcez, and S. Dixon. A hybrid recurrent neural network for music transcription. In *Proceedings of the 40th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 2061–2065, Brisbane, Australia, Apr. 2015. doi:10.1109/ICASSP.2015.7178333. URL http://www.eecs.qmul.ac.uk/~simond/pub/2015/Sigtia-et-al-ICASSP2015.pdf. Preprint http://arxiv.org/abs/1411.1623.

C. N. Silla Jr., A. L. Koerich, and C. A. A. Kaestner. The latin music database. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, pages 451–456, Philadelphia, PA, USA, Sept. 2008. URL http://ismir2008.ismir.net/papers/ISMIR2008_106.pdf.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015. URL http://arxiv.org/abs/1409.1556.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop of the 2nd International Conference on Learning Representations (ICLR)*, Banff, Canada, Apr. 2014. URL http://arxiv.org/abs/1312.6034.

A. J. Simpson, G. Roma, and M. D. Plumbley. Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network. In *Proceedings of the 12th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 429–436, Liberec, Czech Republic, Aug. 2015. doi:10.1007/978-3-319-22482-4_50. Preprint http://arxiv.org/abs/1504.04658.

M. Slaney, Y. Lifshits, and J. He. Optimal parameters for locality-sensitive hashing. *Proceedings of the IEEE*, 100(9):2604–2623, 2012. doi:10.1109/JPROC.2012.2193849. URL http://slaney.org/malcolm/yahoo/Slaney2012(OptimalLSH).pdf.

P. Smaragdis and J. C. Brown. Non-negative matrix factorization for polyphonic music transcription. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 177–180, New Paltz, NY, USA, Oct. 2003. doi:10.1109/ASPAA.2003.1285860. URL http://paris.smaragd.is/pubs/smaragdis-waspaa03.pdf.

J. B. L. Smith, J. A. Burgoyne, I. Fujinaga, D. De Roure, and J. S. Downie. Design and creation of a large-scale database of structural annotations. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 555–560, Miami, FL, USA, Oct. 2011. URL http://ismir2011.ismir.net/papers/PS4-14.pdf.

P. Smolensky. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, pages 194–281. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. doi:104279.104290.

R. Sonnleitner, B. Niedermayer, G. Widmer, and J. Schlüter. A simple and effective spectral feature for speech detection in mixed audio signals. In *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx)*, York, UK, Sept. 2012. URL http://jan-schlueter.de/pubs/2012_dafx2.pdf.

J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *Workshop of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015. URL http://arxiv.org/abs/1412.6806.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

A. M. Stark and M. D. Plumbley. Real-time chord recognition for live performance. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 85–88, Montréal, Canada, 2009. URL http://eecs.qmul.ac.uk/~markp/2009/StarkPlumbley09-icmc.pdf.

S. S. Stevens. The measurement of loudness. *The Journal of the Acoustical Society of America*, 27(5):815–829, Sept. 1955.

S. S. Stevens. On the psychophysical law. *The Psychological Review*, 64(3):153–181, May 1957.

S. S. Stevens and J. Volkmann. The relation of pitch to frequency: A revised scale. *The American Journal of Psychology*, 53(3):329–353, 1940.

I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, GA, USA, June 2013. PMLR. URL http://proceedings.mlr.press/v28/sutskever13.html.

Y. Suzuki, V. Mellert, U. Richter, H. Møller, L. Nielsen, R. Hellman, K. Ashihara, K. Ozawa, and H. Takeshima. Precise and full-range determination of two-dimensional equal loudness contours. Technical Report IS-01, Tohoku University, Japan, 2003.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*,

pages 1–9, Boston, MA, USA, June 2015. doi:10.1109/CVPR.2015.7298594. URL http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf.

C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv e-prints*, abs/1602.07261, 2016a. URL http://arxiv.org/abs/1602.07261.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, Las Vegas, NV, USA, June 2016b. doi:10.1109/CVPR.2016.308. URL http://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.pdf.

M. Tervaniemi and K. Hugdahl. Lateralization of auditory-cortex functions. *Brain Research Reviews*, 43(3):231–246, 2003. doi:10.1016/j.brainresrev.2003.08.004.

J. Thickstun, Z. Harchaoui, and S. Kakade. Learning features of music from scratch. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, Toulon, France, Apr. 2017. URL http://arxiv.org/abs/1611.09827.

T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 1064–1071, Helsinki, Finland, 2008. doi:10.1145/1390156.1390290. URL http://www.cs.toronto.edu/~tijmen/pcd/pcd.pdf.

T. Tieleman and G. E. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 1033–1040, Montréal, Quebec, Canada, 2009. doi:10.1145/1553374.1553506. URL http://www.cs.toronto.edu/~tijmen/fpcd/fpcd.pdf.

J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 648–656, June 2015. doi:10.1109/CVPR.2015.7298664. URL http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Tompson_Efficient_Object_Localization_2015_CVPR_paper.pdf.

A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proceedings of the 21st IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Anchorage,

AK, USA, June 2008. doi:10.1109/CVPR.2008.4587633. URL http://people.csail.mit.edu/torralba/publications/cvpr2008.pdf.

H. Traunmüller. Analytical expressions for the tonotopic sensory scale. *The Journal of the Acoustical Society of America*, 88(1):97–100, 1990. doi:10.1121/1.399849.

D. Turnbull, G. R. G. Lanckriet, E. Pampalk, and M. Goto. A supervised approach for detecting boundaries in music using difference features and boosting. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, pages 51–54, Vienna, Austria, Sept. 2007. URL http://ismir2007.ismir.net/proceedings/ISMIR2007_p051_turnbull.pdf.

G. Tzanetakis and P. R. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002. doi:10.1109/TSA.2002.800560. URL http://www.cs.uvic.ca/~gtzan/work/pubs/tsap02gtzan.pdf.

K. Ullrich, J. Schlüter, and T. Grill. Boundary detection in music structure analysis using convolutional neural networks. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, Taipei, Taiwan, Oct. 2014. URL http://jan-schlueter.de/pubs/2014_ismir.pdf.

K. Ullrich, E. Meeds, and M. Welling. Soft weight-sharing for neural network compression. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, Toulon, France, Apr. 2017. URL http://arxiv.org/abs/1702.04008.

S. Umesh, L. Cohen, and D. Nelson. Fitting the mel scale. In *Proceedings of the 24th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 217–220, Phoenix, AZ, USA, Mar. 1999. doi:10.1109/ICASSP.1999.758101.

A. van den Oord, S. Dieleman, and B. Schrauwen. Transfer learning by supervised pre-training for audio-based music classification. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, pages 29–34, Taipei, Taiwan, Oct. 2014. URL http://www.terasoft.com.tw/conf/ismir2014/proceedings/T007_118_Paper.pdf.

A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv e-prints*, abs/1609.03499, Sept. 2016a. URL http://arxiv.org/abs/1609.03499.

A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1747–1756, New York City, NY, USA, June 2016b. PMLR. URL http://proceedings.mlr.press/v48/oord16.html.

B. Videt. Turkish van cat, 2006. URL http://commons.wikimedia.org/wiki/File:Turkish_Van_Cat.jpg. Licensed under CC BY 2.5 (http://creativecommons.org/licenses/by/2.5).

E. Vincent, R. Gribonval, and C. Févotte. Performance measurement in blind audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1462–1469, July 2006. doi:10.1109/TSA.2005.858005. URL http://hal.archives-ouvertes.fr/inria-00544230.

O. Vinyals and D. Povey. Krylov subspace descent for deep learning. In N. D. Lawrence and M. Girolami, editors, *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 22 of *Proceedings of Machine Learning Research*, pages 1261–1268, La Palma, Canary Islands, Apr. 2012. PMLR. URL http://proceedings.mlr.press/v22/vinyals12.html.

A. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal processing*, 37(3):328–339, Mar. 1989. doi:10.1109/29.21701. URL http://www.cs.toronto.edu/~hinton/absps/waibelTDNN.pdf.

G. H. Wakefield. Mathematical representation of joint time-chroma distributions. In *Advanced Signal Processing Algorithms, Architectures, and Implementations IX (Proceedings SPIE)*, volume 3807, pages 637–645, 1999. doi:10.1117/12.367679.

E. H. Weber. Der Tastsinn und das Gemeingefühl. In R. Wagner, editor, *Handwörterbuch der Physiologie mit Rücksicht auf physiologische Pathologie*, volume 3, part 2, pages 481–588. Friedrich Vieweg und Sohn, Braunschweig, Germany, 1846. URL http://echo.mpiwg-berlin.mpg.de/MPIWG:TUHHCUVC.

R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, pages 194–205, New York City, NY, USA, Aug. 1998. URL http://www.vldb.org/conf/1998/p194.pdf.

J. Weston, S. Bengio, and P. Hamel. Multi-tasking with joint semantic spaces for large-scale music annotation and retrieval. *Journal of New Music Research*,

40(4):337–348, 2011. doi:10.1080/09298215.2011.603834. URL http://bengio.abracadoudou.com/cv/publications/pdf/weston_2011_jnmr.pdf.

P. M. Williams. Using neural networks to model conditional multivariate densities. *Neural Computation*, 8(4):843–854, 1996. doi:10.1162/neco.1996.8.4.843.

L. Xie, J. Wang, Z. Wei, M. Wang, and Q. Tian. DisturbLabel: Regularizing CNN on the loss layer. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4753–4762, Las Vegas, NV, USA, June 2016. doi:10.1109/CVPR.2016.514. URL http://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Xie_DisturbLabel_Regularizing_CNN_CVPR_2016_paper.pdf.

F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016. URL http://arxiv.org/abs/1511.07122.

H. Yu, Z. H. Tan, Y. Zhang, Z. Ma, and J. Guo. DNN filter bank cepstral coefficients for spoofing detection. *IEEE Access*, 5:4779–4787, Mar. 2017. ISSN 2169-3536. doi:10.1109/ACCESS.2017.2687041. Preprint http://arxiv.org/abs/1702.03791.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Proceedings of the 13th European Conference on Computer Vision (ECCV)*, volume 1, pages 818–833, Zürich, Switzerland, Sept. 2014. Springer International Publishing. URL http://www.cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf. Preprint http://arxiv.org/abs/1311.2901.

V. Zenz and A. Rauber. Automatic chord detection incorporating beat and key detection. In *Proceedings of the IEEE International Conference on Signal Processing and Communications (ICSPC)*, pages 1175–1178, Dubai, United Arab Emirates, Nov. 2007. doi:10.1109/ICSPC.2007.4728534. URL http://publik.tuwien.ac.at/files/pub-inf_5274.pdf.

Z.-H. Zhou and M.-L. Zhang. Neural networks for multi-instance learning. Technical report, Nanjing University, China, Aug. 2002. URL http://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/techrep02.pdf.

Y. Zhu, M. S. Kankanhalli, and S. Gao. Music key detection for musical audio. In *Proceedings of the 11th International Conference on Multimedia Modeling (MMM)*, pages 30–37, Jan. 2005. doi:10.1109/MMMC.2005.56.

Y. Zhu, Q. Sun, and S. Rahardja. Detecting musical sounds in broadcast audio based on pitch tuning analysis. In *Proceedings of the 7th IEEE International Conference on Multimedia and Expo (ICME)*, pages 13–16, Toronto, Canada, July 2006. doi:10.1109/ICME.2006.262502. URL `http://www.ee.columbia.edu/~qibin/papers/qibin2006_icme_3.pdf`.

E. Zwicker. Subdivision of the audible frequency range into critical bands (Frequenzgruppen). *The Journal of the Acoustical Society of America*, 33(2):248–248, 1961. doi:10.1121/1.1908630.

Curriculum Vitae of the Author

Personal Data

Name:	Jan Schlüter
Date of birth:	October 1, 1985
Place of birth:	Elmshorn, Germany
Email:	jan@jan-schlueter.de
Website:	www.jan-schlueter.de

Education

10/2011–09/2017	PhD in Informatics at the Johannes-Kepler University Linz. *Thesis:* "Deep Learning for Event Detection, Sequence Labelling and Similarity Estimation in Music Signals"
10/2008–10/2011	Master of Science in Informatics at the Technical University of Munich. *Thesis:* "Unsupervised Audio Feature Extraction for Music Similarity Estimation"
09/2009–12/2009	Erasmus student at the University of Helsinki and TKK Helsinki
10/2005–10/2008	Bachelor of Science in Informatics at the University of Hamburg. *Thesis:* "Accelerating the Debugging Process within a Development Environment for Multi-Agent Systems"

Experience

since 08/2014	Maintainer and core developer of open source project Lasagne
since 08/2013	Maintainer of open source project cudamat
since 05/2011	Research assistant at the Austrian Research Institute for Artificial Intelligence (OFAI). *Project:* Automatic Segmentation, Labelling, and Characterisation of Audio Streams. *Additional responsibilities:* Commercializing research results with industry partners (e.g., B&O, RadioAnalyzer, SWISSPERFORM)

02/2008–08/2008	Tutor for Human-Computer Interaction (University of Hamburg)
10/2007–02/2008	Tutor for Logic Programming (University of Hamburg)
04/2007–09/2007	Student assistant at the international graduate research group CINACS (University of Hamburg)
since 05/2005	Operating a business developing and selling software for finding duplicate files
12/2004–08/2005	Community service at a youth-educational institution in Barmstedt, Germany

Awards and Prizes

08/2016	Best oral presentation award at the 17th International Society for Music Information Retrieval Conference (ISMIR) for the paper "Learning to pinpoint singing voice detection from weakly labeled examples" by J. Schlüter
10/2012	Best paper award at the 13th International Society for Music Information Retrieval Conference (ISMIR) for the paper "A MIREX meta-analysis of hubness in audio music similarity" by A. Flexer, D. Schnitzer, and J. Schlüter
2008–2011	Granted a scholarship by the German National Academic Foundation (Studienstiftung des deutschen Volkes)

Scientific Services

Reviewer for international journals:

> IEEE/ACM Transactions on Audio, Speech and Language (2013, 2014, 2015, 2016), IEEE Signal Processing Letters (2013), IEEE Transactions on Systems, Man and Cybernetics (2016), International Journal of Multimedia Information Retrieval (2014)

Reviewer for international conferences:

> AAAI (2015), ACMMM (2012), CBMI (2016), ECML (2012, 2013), EUSIPCO (2013, 2014), ICMC (2012), ISMIR (2012, 2013, 2014, 2015, 2016, 2017)

Publications

Publications forming the main chapters of the thesis (■), publications from side projects and collaborations next to or before the thesis work (•), workshop contributions and technical reports (◦):

- T. Grill and J. Schlüter. Two convolutional neural networks for bird detection in audio signals. In *Proceedings of the 25th European Signal Processing Conference (EUSIPCO)*, Kos Island, Greece, Aug. 2017. URL http://jan-schlueter.de/pubs/2017_eusipco.pdf. To appear.

◦ M. Dorfer, J. Schlüter, A. Vall, F. Korzeniowski, and G. Widmer. End-to-end cross-modality retrieval with CCA projections and pairwise ranking loss. *arXiv e-prints*, abs/1705.06979, May 2017. URL http://arxiv.org/abs/1705.06979.

- S. Böck, F. Korzeniowski, J. Schlüter, F. Krebs, and G. Widmer. madmom: a new python audio and music signal processing library. In *Proceedings of the 24th ACM International Conference on Multimedia (ACMMM)*, pages 1174–1178, Amsterdam, Netherlands, Oct. 2016. doi:10.1145/2964284.2973795. URL http://www.cp.jku.at/research/papers/Boeck_etal_ACMMM_2016.pdf.

■ J. Schlüter. Learning to pinpoint singing voice from weakly labeled examples. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, New York City, NY, USA, Aug. 2016. URL http://jan-schlueter.de/pubs/2016_ismir.pdf. Best oral presentation award.

◦ R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, et al. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL http://arxiv.org/abs/1605.02688. Project URL http://github.com/Theano/Theano.

■ J. Schlüter and T. Grill. Exploring data augmentation for improved singing voice detection with neural networks. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, Málaga, Spain, Oct. 2015. URL http://jan-schlueter.de/pubs/2015_ismir.pdf.

- T. Grill and J. Schlüter. Music boundary detection using neural networks on combined features and two-level annotations. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*,

Málaga, Spain, Oct. 2015b. URL http://jan-schlueter.de/pubs/2015_ismir2.pdf.

- T. Grill and J. Schlüter. Music boundary detection using neural networks on spectrograms and self-similarity lag matrices. In *Proceedings of the 23rd European Signal Processing Conference (EUSIPCO)*, Nice, France, Aug. 2015a. URL http://jan-schlueter.de/pubs/2015_eusipco.pdf.

■ K. Ullrich, J. Schlüter, and T. Grill. Boundary detection in music structure analysis using convolutional neural networks. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, Taipei, Taiwan, Oct. 2014. URL http://jan-schlueter.de/pubs/2014_ismir.pdf.

■ J. Schlüter and S. Böck. Improved musical onset detection with convolutional neural networks. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6979–6983, Florence, Italy, May 2014. doi:10.1109/ICASSP.2014.6854953. URL http://jan-schlueter.de/pubs/2014_icassp.pdf.

○ J. Schlüter. Restricted Boltzmann machine derivations. Technical Report TR-2014-13, Österreichisches Forschungsinstitut für Artificial Intelligence (OFAI), Vienna, Austria, Mar. 2014. URL http://jan-schlueter.de/pubs/2014_techrep_rbm.pdf.

■ J. Schlüter. Learning binary codes for efficient large-scale music similarity search. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, Curitiba, Brazil, Nov. 2013. URL http://jan-schlueter.de/pubs/2013_ismir.pdf.

○ J. Schlüter and S. Böck. Musical onset detection with convolutional neural networks. In *6th International Workshop on Machine Learning and Music (MML), in conjunction with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, Prague, Czech Republic, Sept. 2013. URL http://jan-schlueter.de/pubs/2013_mml.pdf.

- D. Schnitzer, A. Flexer, and J. Schlüter. The relation of hubs to the Doddington zoo in speaker verification. In *Proceedings of the 21st European Signal Processing Conference (EUSIPCO)*, Marrakech, Morocco, Sept. 2013. URL http://jan-schlueter.de/pubs/2013_eusipco.pdf.

- B. Ionescu, J. Schlüter, I. Mironică, and M. Schedl. A naïve mid-level concept-based fusion approach to violence detection in hollywood movies. In *Proceedings of the ACM International Conference on Multimedia Retrieval (ICMR)*, Dallas, TX, USA, Apr. 2013. URL http://jan-schlueter.de/pubs/2013_icmr.pdf.

  - J. Schlüter, B. Ionescu, I. Mironică, and M. Schedl. ARF @ MediaEval 2012: an uninformed approach to violence detection in hollywood movies. In *MediaEval 2012 Workshop*, Pisa, Italy, Oct. 2012. URL http://jan-schlueter.de/pubs/2012_mediaeval.pdf.

  - B. Ionescu, I. Mironică, K. Seyerlehner, P. Knees, J. Schlüter, M. Schedl, H. Cucu, A. Buzo, and P. Lambert. ARF @ MediaEval 2012: multimodal video classification. In *MediaEval 2012 Workshop*, Pisa, Italy, Oct. 2012. URL http://jan-schlueter.de/pubs/2012_mediaeval2.pdf.

- A. Flexer, D. Schnitzer, and J. Schlüter. A MIREX meta-analysis of hubness in audio music similarity. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR)*, Porto, Portugal, Oct. 2012. URL http://jan-schlueter.de/pubs/2012_ismir.pdf. Best paper award.

  - J. Schlüter and R. Sonnleitner. Unsupervised feature learning for speech and music detection in radio broadcasts. In *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx)*, York, UK, Sept. 2012. URL http://jan-schlueter.de/pubs/2012_dafx.pdf.

- R. Sonnleitner, B. Niedermayer, G. Widmer, and J. Schlüter. A simple and effective spectral feature for speech detection in mixed audio signals. In *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx)*, York, UK, Sept. 2012. URL http://jan-schlueter.de/pubs/2012_dafx2.pdf.

- J. Schlüter and C. Osendorfer. Music similarity estimation with the mean-covariance restricted Boltzmann machine. In *Proceedings of the 10th International Conference on Machine Learning and Applications (ICMLA)*, Honolulu, HI, USA, Dec. 2011. URL http://jan-schlueter.de/pubs/2011_icmla.pdf.

- J. Schlüter. Unsupervised audio feature extraction for music similarity estimation. Master's thesis, Technische Universität München, Munich, Germany, Oct. 2011. URL http://jan-schlueter.de/pubs/msc.pdf.

- C. Osendorfer, J. Schlüter, J. Schmidhuber, and P. van der Smagt. Unsupervised learning of low-level audio features for music similarity estimation. In *ICML Workshop on Learning Architectures, Representations, and Optimization for Speech and Visual Information Processing*, Bellevue, WA, USA, June 2011. URL http://jan-schlueter.de/pubs/2011_icmlws.pdf.
- J. Schlüter, L. Cabac, and D. Moldt. Adding runtime net manipulation features to MulanViewer. In *15th German Workshop on Algorithms and Tools for Petri Nets (AWPN)*, Rostock, Germany, Sept. 2008. URL http://ceur-ws.org/Vol-380/paper14.pdf.