University of Amsterdam (UVA)

Austrian Research Institute for Artificial Intelligence (OFAI)

Faculty of Science

MASTER OF SCIENCE THESIS

KAREN ULLRICH

FEED-FORWARD NEURAL NETWORKS FOR BOUNDARY DETECTION IN MUSIC STRUCTURE ANALYSIS

SUPERVISOR:
Prof. Gerhard Widmer
Prof. Max Welling

Vienna, August 28, 2014

DECLARATION OF AUTHORSHIP

I hereby declare and confirm that this thesis is entirely the result of my own work except where otherwise indicated.

. . . . . .

Signature

At this point, I'd like to thank everybody that supported this work.
In particular, I thank **Prof. Widmer** for providing the topic and aid. **Prof. Welling** for thoughts and inputs to the experimental work and supervision at UvA. Especially, I want to thank my local supervisors **Jan Schlüter** and **Dr. Thomas Grill** for numerous hours of support in every respect. Finally, I thank the **OFAI** for the admittance and granting.
This research was conducted in the context of the project 'Automatic Segmentation, Labelling, and Characterisation of Audio Streams' (project number TRP 307-N23), funded by the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit) and Austrian Science Fund (FWF).

Abstract

Following pioneering studies that first applied neural networks in the field of music information retrieval (MIR), we apply feed forward neutral networks to retrieve boundaries in musical pieces, e.g., between chorus and verse. Detecting such segment boundaries is an important task in *music structure analysis*, a sub-domain of MIR. To that end, we developed a framework to perform supervised learning on a representative subset of the SALAMI data set, containing structural annotations. More specifically, we apply convolutional networks to learn spatial relationships and fully connected layers, to detect segment boundaries automatically. In that context, the data was presented to the networks as mel-scaled magnitude spectrograms. Furthermore, we applied the dropout technique. After optimising our models with respect to various hyper-parameters, we find them to outperform the F-score of any algorithm in the MIREX campaign of 2012 and 2013. In particular, we achieved F-measures of 0.476 for tolerances of $\pm 0.5s$ and 0.619 for tolerances of $\pm 3s$. These are differences to current techniques of 0.14 and 0.09. Our method is particularly outstanding because it is mainly data driven and does not utilize hand-crafted high-level features to create classifiers. When investigating the method further, we find that even with such a simple general-purpose feature as chroma vectors and no convolutional layers we can still achieve results comparable to existing algorithms. Moreover, we visualised which regions of the input are of highest interest for our networks. As a result, we found all networks to concentrate on very similar time and frequency bands.

Contents

1. Introduction

Understanding and imitating human perception with computational means is a classic problem in Artificial Intelligence. For example, computer vision (CV), speech recognition (SR) and music information retrial (MIR) attempt to model human vision, speech and music understanding respectively. In CV and SR, state-of-the-art methods could not improve the performance on benchmark data sets significantly for many years [50, 1]. In the early 10's, however, neural networks (NNs) boosted relevant scores in both disciplines considerably [18, 41]. Despite their successful application in these two fields, NNs are not applied frequenly in MIR, yet. Pioneering work in this field has been conducted by Schlüter and Böck [72] and Boulanger-Lewandowski et al. [12] that utilised NNs for onset detection and chord recognition (sub-domains of MIR). Their methods scored well on relevant data sets.

Considering these promising results, we will explore whether NNs are also applicable to *Music Structure Analysis* (MSA), another sub-branch of MIR, in this study. MSA refers to the task of recovering a description of the sectional form of a piece of music, e.g., chorus and verse [64]. Note that the structure of a musical piece is typically hierarchical as is illustrated in Figure 1.1. Furthermore, we want to point out, that there is high inter-individual variability in the nature of music perception, to the effect that retrieving musical structure can be ambiguous. Even with annotation guidelines at hand, trained human annotators have a high disagreement rate regarding their analysis. Smith [77] examined this for pieces of experimental music as illustrated by Figure 1.2. Here one piece of music is reviewed by two experts. When comparing the temporal points separating structural elements, the so called *musical segment boundaries* (green lines), we recognise two phenomena. For one, there is the temporal inaccuracy of boundaries. Two human annotators hardly ever predict boundaries at the exact same time. In order to account for that issue, we will introduce a binary measure for boundary agreement later, that relies on a certain time tolerance for boundaries to be in agreement. Second, the number of segments and the hierarchical classification depth seems to be controversial from expert to expert. Since this temporal uncertainty of targets is not existent in problems of CV, we aim to develop a new way to capture given uncertainties in our model. To that end, we will develop a model that retrieves the boundaries between the main structural parts of a piece of music given annotated data.

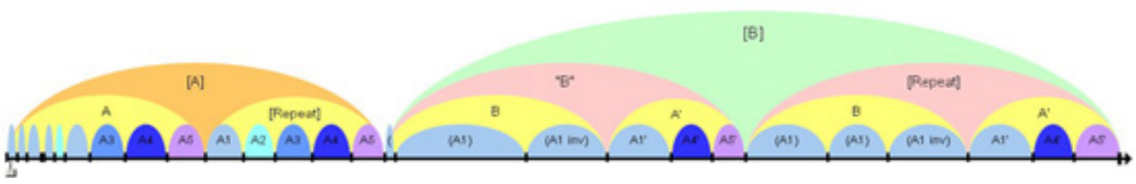

Figure 1.1: The structure of a piece of music has been visualised by an arc diagram. The abscissa relates to the ongoing time. Figure from SALAMI-Blog [71].

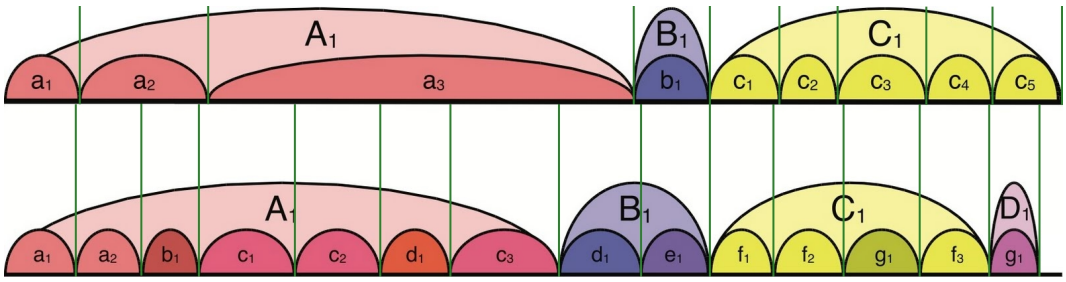

Figure 1.2: One piece of experimental music annotated by different experts. Musical boundaries are indicated by green lines. Figure from Smith [77].

Following we will lay out the goals of this work explicitly:

(1) The study was initiated because we expect frequency-time relationships in music to be highly comparable to spatial relationships in pictures for which NN models have been extensively developed in the past years. Thus, the first objective of this study is to find a suitable representation of music to examine our hypothesis that NNs can identify structure in audio.

(2) Given suitable features we strive to develop and train a data driven model. In particular that means finding a way to present the data and the annotated boundaries to an ML classifier and exploring how to adjust standard methods to optimise model parameters given the properties of our data named above.

(3) In order to gain insights into our models, we aim to visualise certain aspects of their behaviour for inference of what the model learned.

(4) Furthermore, we want to show that enabling the model to learn the representation of the data by itself (i.e., applying convolutional layers) outperforms models that use hand-designed features (i.e., chroma vectors).

Our efforts will result in a model that sets new standards beating current state-of-the-art algorithms considerably [84]. Moreover, we were able to identify the temporal context most important for predicting a musical boundary as well as the most important frequencies contributing to the decisions of our classifiers.

In the second chapter, this thesis will give an overview on data representations in MIR and consequently on techniques currently applied in MSA. Subsequently, we convey a theoretical background of NN models and their training. Finally, this chapter will close with a description of possible model evaluations. In the third chapter, we will explain the method applied in this study. In the last chapters we will present and discuss experiments we conducted. We will finish this work by a discussion and an outlook on future research.

Please note that written consent has been expressed by co-authors Jan Schlüter and Thomas Grill for the partial use of contents published in the joint paper Ullrich et al. [84]. This especially applies to Section 2.3.2 and 3.1, which have essentially been quoted.

2. Background

In this chapter we will present current music structure analysis (MSA) methods and introduce a new data-driven model for the task. For that purpose, we will provide theoretical background on music related feature extraction. We will furthermore review current music information retrieval (MIR) techniques for MSA and finally describe feed-forward neural networks (FNN), a supervised non-linear data-based model.

2.1. Feature extraction and common methods

Analysing music is a demanding task, because the experience of listening to it is highly subjective and rarely do two human beings share the exact same perception when listening to a piece. Thus, capturing that by algorithms is extremely challenging. However, even untrained listeners tend to organise perceived acoustic information into hierarchies and structures with regard to various musical aspects, e.g., identifying recurrent themes or detecting temporal boundaries between contrasting musical parts. In this section, we will give a small overview of methods for computational music structure analysis. The main goal is to provide a selection of features. These will later be processed by machine learning techniques to divide an audio recording into temporal segments corresponding to musical parts such as chorus or verse. Specifically, we address different musical dimensions such as melody, harmony, rhythm, and timbre. Additionally, we review techniques from the field of MIR that try to tackle the segmentation problem. Finally, we will conclude with what we have learned from current MIR research and how this affects our work.

2.1.1. Features and motivation

A digital audio signal comes as a sampled waveform. It is relatively uninformative to the eye by itself. Furthermore, the amount of data is a problem for algorithms. Thus, audio pre-processing needs to be employed. In particular, we are concerned with finding features (mappings) that relate to human perception. In that context, we will explore the musical dimensions mentioned above and relate features to them. Bruderer et al. [13] studied perceptual cues humans use to determine segmentation points in music. The results of their work indicates that "global structure", "change in timbre", "change in level (loudness)", "repetition", and "change in rhythm" mark the presence of a structural boundaries to listeners. Following, we will discuss manifold mappings representing these perceptional cues.

Mel-frequency cepstral coefficients (MFCCs) MFCCs aim to capture the timbre or the instrumentation of a piece of music. The timbre, a psychoacoustic measure also known as the tone color, is a quality of any single sound and based on the frequency spectrum. "Perceptually, timbre is closely related to the recognition of sound

sources and depends on the relative levels of the sound at critical bands[1] as well as their temporal evolution" [64]. This is why, instead of directly applying a linear frequency spectrum, it is common for many timbre-based structure analysis methods to utilise MFCCs. This signal representation is a mel-scaled [2] frequency spectrum adjusted to the anatomy of the human ear. MFCCs are obtained by calculating the discrete cosine transformed (DCT) log-power spectrum on the mel frequency scale

$$\text{MFCC}(k) = \sum_{b=0}^{N-1} E(b) \cos\left(\frac{\pi(2b+1)k}{2N}\right), \tag{2.1}$$

in which k denotes an index, b denotes the subbands (frequency bands) which are uniformly distributed over the mel- frequency scale and E(b) denotes the corresponding log-energy. Note that MFCCs corresponding to lower frequencies are more closely related to the aspect of timbre than higher ones [64]. For an extended, more technical, discussion about computing MFCCs see Logan [49].

Chroma A second important feature to describe harmonic and melodic sequences in the context of music structure analysis is chroma, also called pitch class profiles. Chroma aim to correspond to the set $\{C, C\#, D, ..., B\}$ containing the 12 pitch classes of Western music notation. A chroma vector then describes how the spectral energy $E(b)$ of a signal is distributed among these 12 pitch classes while ignoring octave information. This mapping turns out to be a powerful representation of the harmonic aspect of music [8, 30, 15, 51, 53]. Computationally, many algorithms for calculating chroma-based audio features have been proposed. The majority of the approaches compute a discrete Fourier transform (DFT) first, and subsequently pool the DFT coefficients into chroma bins [8, 30, 33]. For a more detailed description and more advanced approaches see Gómez [30] and Müller et al. [57][53, 54].

Onsets In contrast to timbre and harmonic features, beat, tempo or rhythm based approaches are usually not stand-alone but rather support chroma and MFCCs by adding temporal information. This can improve the precision of found segment boundaries [45, 9, 89]. In order to extract tempo information from audio recordings, it is common to locate so called onsets. Onsets correlate with positions of note onsets in music. In popular music this mostly correlates with the current beat. In other genres like jazz we can not find such a generalisation. In order to find onsets in an audio signal, onset detection functions are developed. They typically analyse sudden changes of signal energy and spectrum [9, 89, 72]. The agreement of recent state-of-the-art algorithms with annotations is very high. Thus, with such precise onsets, one can proceed to finding quasi-periodic patterns on the detected onsets. Important for the analysis is to obtain a shift-invariant spectrogram that is immune to the exact temporal position of the pattern. Approaches for that range from autocorrelation-based approaches [19, 67] to omitting phase information via short-time Fourier transforms [34, 67]. Visualisations of temporal information are referred to as tempogram [14], rhythmogram [39], or beat spectrogram [25].

Spectrograms, MFCCs, chroma vectors and onsets represent the basis for other analysis methods in MIR. Paulus et al. [65] provide a visual overview (see Figure 2.1). The reader can compare the annotated ground truth with patterns in given features.

[1] According to psychoacoustics, critical bands are frequency bands that relate to human perception. Specifically, a critical band describes a frequency bandwidth in which the perception of two tones can not be resolved.

[2] Linear for low frequencies and logarithmic for higher frequencies.

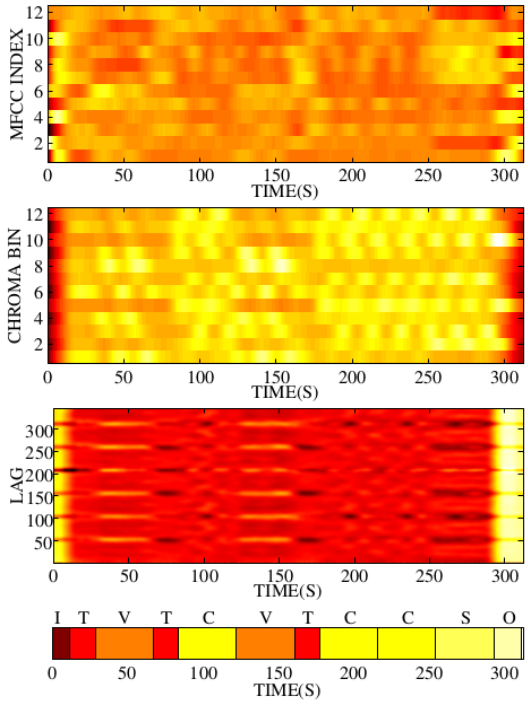

Figure 2.1: The piece “Tuonelan koivut” by the Finish heavy metal band Kotiteollisuus depicted as MFCCs (first panel), chroma (second panel), and rhythmogram (third panel). The song is represented by the given features. One may make out certain correlations with the annotated musical parts with the bare eye .Here, the abbreviations I,T,V,C,S,O denote intro, theme, verse, chorus, solo, and outro respectively. Figure by Paulus et al. [64].

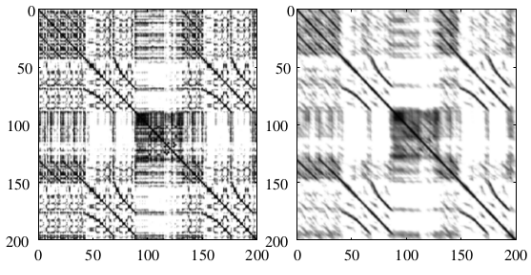

Figure 2.2: A piece of music with repeated parts in different tempi is represented by an SDM. The two SDMs differ in feature resolution. Note that the dark lines indicating high similarity are more visible in the lower resolution SDM (right). Furthermore, curves “parallel” to the main diagonal can be identified as repeating parts. Figure by Paulus et al. [64].

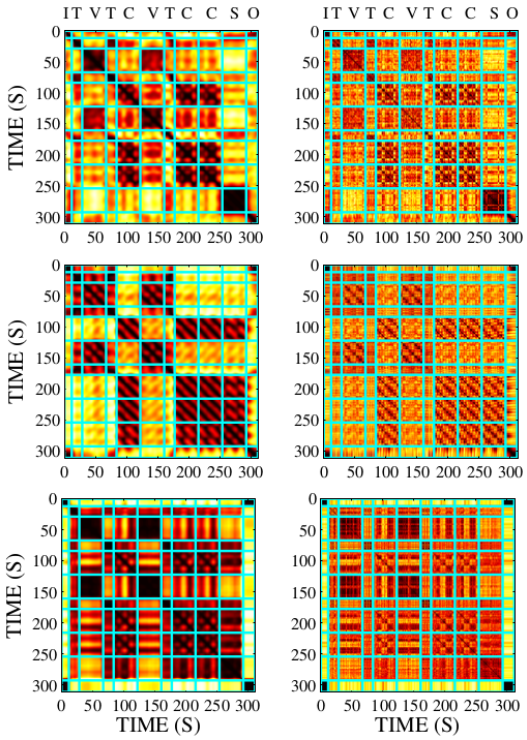

Figure 2.3: The features from Figure 2.1 are used to create SDMs with coarse (left) and fine (right) time scale. Top: MFCCs. Middle: Chroma features. Bottom: Rhythmogram. The annotated structure of the piece is indicated by the blue grid. Note how the segment boundaries are more clear in this representation than in Figure 2.1 and how different features share perceptual aspects but not all. The ground truth is indicated by labels at the top of the figure. Figure by Paulus et al. [64].

Self distance matrices (SDMs) SDMs are common in many sciences to detect pattern repetition. Given a sequence of data vectors $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$, the general idea is to compare each point $\mathbf{x}_i$ with every point $\mathbf{x}_j$ with the help of a distance measure $d(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}^+$. The SDM is obtained by computing the $N \times N$ matrix $\mathbf{D}_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)\ \forall i, j \in \{1, 2, ..., N\}$. Equivalently, one can specify a self-similarity matrix (SSIM) correlating to a given distance measure by $\mathbf{S}_{ij} = 1 - d(\mathbf{x}_i, \mathbf{x}_j)\ \forall i, j \in \{1, 2, ..., N\}$. It is important to remark that in this definition it is assumed that $d(.,.) \in [0, 1]$. In MIR, SDMs have been introduced by Foote [23]. The objective is to observe the time structure of an audio recording at hand. Frequently used distance measures in this discipline comprise the Euclidean or L^2-norm $d_E(\mathbf{x}_i, \mathbf{x}_j) = ||\mathbf{x}_i - \mathbf{x}_j||$, and the cosine distance

$$d_C(\mathbf{x}_i, \mathbf{x}_j) = 0.5 \left(1 - \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{||\mathbf{x}_i|| ||\mathbf{x}_j||}\right) \tag{2.2}$$

in which $\langle ., . \rangle$ denotes the dot product and $||.||$ a vector norm. Typically, the distance measure is symmetric, resulting in a likewise symmetric matrix. This makes $N(N-1)/2$ points of the matrix redundant. Usually in MIR, distance measures are designed so that they compare single time frames of features. This, however, can lead to discontinuities. To remove them, Foote [23] proposed a method to smooth the SDMs by averaging the distance values from a number of neighbouring frames and to utilise that as the distance measure. Other approaches include calculating the average distance from feature vectors within non-overlapping segments [73, 61]. Another way is to compute SDMs with different temporal hierarchical levels, i.e., ranging from individual frames to musical patterns while each SDM of higher level is calculated with SDMs of lower structure. The main difficulty for MIR applications is to find a fitting data representation to create an expressive SDM. Frequently, chroma vectors are used for data representation since they account for the fact that themes are often repeated in another key [33, 53]. More advanced approaches may include multiple features in the representation vector. Interestingly, the feature resolution or more general temporal parameters may play a role in the occurrence of patterns in the SDM, apart from the appropriate feature choice alone [66]. Thus, working with low resolution may be beneficial with respect to computational costs and for structural reasons, although one might lose precision [53, 62]. This behaviour is demonstrated in Figure 2.2. In SDMs the occurrence of two phenomena leads to inference about the musical structure of a given recording. On one hand, blocks of high similarity (low distance) are formed when musical properties such as instrumentation stay constant over the duration of a musical part. On the other hand, one observes stripes parallel to the main diagonal when sequences of a piece are repeated. For an illustration see Figure 2.3.

2.1.2. Review on MIR structure analysis

This section will provide a short review about structure analysis methods in MIR that are currently common. All presented methods are based on the features we described in the previous section. The literature divides the various approaches into three categories: repetition-based, novelty-based and homogeneity-based [64].

Novelty-based approaches Novelty detection aims to automatically locate points of change and high contrast. Those points are commonly attracting the attention of listeners and thus may lead to the perception of segment boundaries. A frequently applied approach was introduced by Foote [24] who applied an $N \times N$ SDM $\mathbf{S}$ that is based on MFCCs due to their indication of timbre or instrumentation change. Furthermore, one could compute chroma or rhythmogram based SDMs to obtain indicators for changes in harmony, rhythm, or tempo. Equivalent to the idea of a covariance matrix, a correlation matrix is computed with the help of a lower dimen-

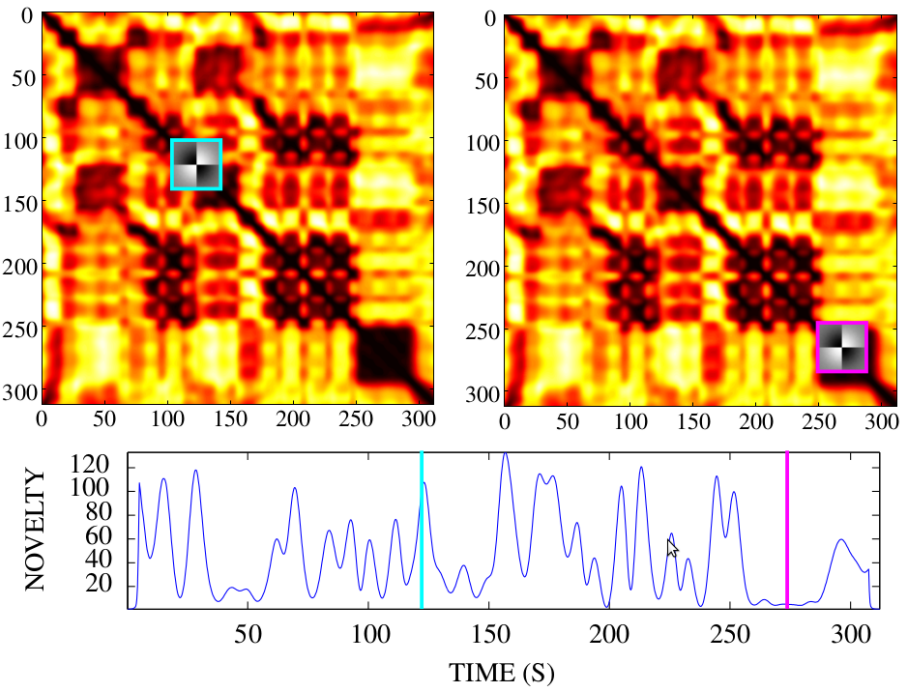

Figure 2.4: Here, the two MFCC-based SDMs from Figure 2.3 are correlated with a checker-board kernel along the main diagonal (top), resulting in a novelty curve (bottom). Two instances are color-coded for better illustration. Figure by Paulus et al. [64].

sional kernel $\mathbf{C} \in \mathbb{R}^{M \times M}, M < N$. This leads to a novelty function n that is designed to detect 2D corner points along the main diagonal of $\mathbf{S}$ which may indicate a segment boundary.

$$n(i) = \sum_{m,n=-L/2}^{L/2} \mathbf{C}(m,n)\mathbf{S}(i+m, i+n) \tag{2.3}$$

in which L denotes the kernel width. Small L will detect novelty on a short term scale whereas large kernels will serve the opposite purpose. In order to find corner points, the kernel has a 2×2 checker-board-like structure and is usually weighted by a Gaussian radial function. Figure 2.4 illustrates an example. One can easily see that the novelty function peaks when similarity changes most on the main diagonal of the audio recording's SDM. Subsequently, to identify segment boundaries we need to detect the peaks of $n(i)$. This task is further described in Chapter 3. Another approach to detect the segment boundaries via an SDM is given by Jensen [39], who applied a similar approach with more complex features. SDMs have long been the centre of this novelty-based research. However, there are first studies indicating the applicability of supervised learning methods in novelty detection which outperform hand-crafted methods. The first supervised machine learning approach was introduced by Turnbull and Lanckriet [83] who combined several features to perform ada-boosting. They did, however, not account for neighbourhood relations of features or temporal context (other than via the derivative). Recently, McFee and Ellis [52] implemented a method based on Fishers linear discriminant to model musical parts. We will review both methods more carefully in Section 2.1.4.

Homogeneity-based approaches Homogeneity-based approaches identify entire segments based on similarities within musical parts. In order to find segment boundaries they often rely on boundary points estimated

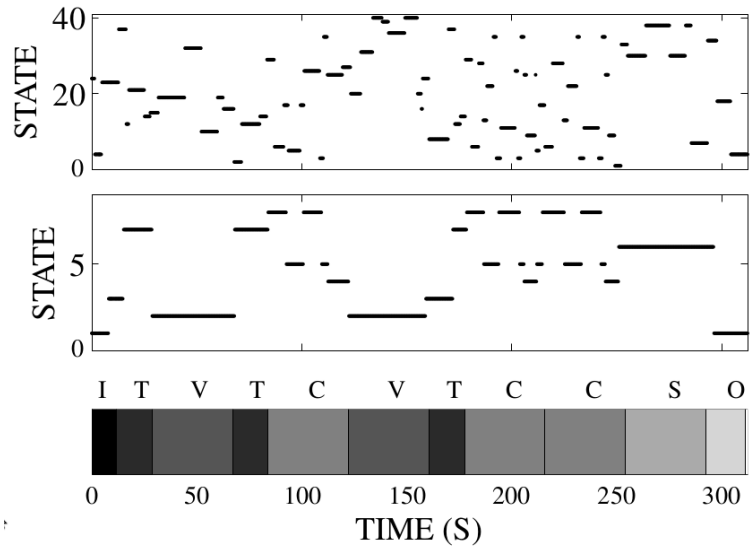

Figure 2.5: State sequences resulting from a fully connected HMM using 40 (top) and 8 (middle) states applied to the MFCC feature sequence of Figure 2.1. The bottom panel shows the annotated ground truth structure. Figure by Paulus et al. [64].

by novelty-based approaches. One homogeneity-based approach uses the segment boundary points obtained through novelty detection. The method introduced by Cooper and Foote [17] aims to find homogeneous clusters regarding acoustic features and thus specifying them. First, the content of each detected segment is modelled by a Normal distribution. Later methods also introduced Gaussian parametrisation [49]. With a probability distribution approximation at hand, the similarity between two segments can be computed by, e.g., the Kullback-Leibler-divergence. For details regarding the divergence measure and the modelling, see Goldberger et al. [28], Cooper and Foote [17] and Weiss [85]. Finally, with a distance map at hand, segments can be grouped with spectral clustering. Similar segments thereby belong to one type of musical part e.g. chorus.
A second important recipe to find homogeneous segments covers the representation of musical parts as hidden Markov model (HMM) states [26] [5]. In an HMM, we can compute the probability of a state sequence $\mathbf{q} = (q_1, q_2, ..., q_N)$ given the observation sequence $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$ by

$$p(\mathbf{q}|\mathbf{X}) \propto P(\mathbf{x}_1|q_1) \prod_{n=2}^{N} P(\mathbf{x}_n|q_n) p(q_n|q_{n-1}) \tag{2.4}$$

in which $P(\mathbf{x}_n|q_n)$ denotes the likelihood of observing $\mathbf{x}_n$ in case the state is q_n, and $p(q_n|q_{n-1})$ denotes the transition probability from state q_{n-1} to q_n. An HMM is trained with the sequence to be analysed. Next, the same sequence is being decoded (modelled) by the HMM. Unfortunately, the HMMs tend to model short term events rather than describing long term musical parts. A visualisation of an 8- and a 40-states HMM is shown in Figure
reffig:hmm. To account for this temporal fragmentation various post-processing methods have been proposed [45]. Grosche et al. [35] introduced the idea of computing the histogram of the states with a sliding window over the sequences. The resulting histogram vectors may consequently be utilised as a feature for probabilistic clustering [5, 46, 45]. A similar alternative to HMMs was introduced by Barrington et al. [7]. The so called dynamic texture mixture seems to create less temporal fragmentation than the model described above.

Repetition-based approaches In contrast to novelty-based approaches and equivalently to homogeneity-based approaches, repetition-based approaches mostly locate entire segments rather than boundary points. As explained in the previous section, repetitive patterns can be found in the SDM by locating stripes parallel to the main diagonal. Although this might seem a trivial task for a human, algorithmic search might be tricky due to distortion in the pattern caused by timbre or rhythmic variabilities or tempo progression [56, 53]. One idea is to enhance the SDM pattern by low-pass filtering to smooth the SDM along the main diagonal [8, 86]. Another way is to average neighbouring bins of the SDM [23]. Furthermore one may compute multiple SDMs with different sliding windows and then combine them via element-wise multiplication to enhance a stable pattern [51]. Those basic ideas have been extended and combined in Peeters [68], Ong [60], Goto [33], Eronen [21], Peeters [67]. Nonetheless, all these methods are somewhat restricted to the assumption that the repetitive parts are all the same tempo. Thus, theoretically all stripes should run exactly parallel to the main diagonal. In practice, this assumption does not hold, e.g., in classical music we will easily find parts repeated in different tempi. This causes arched and even more complex stripes, see Figure 2.2. Müller and Clausen [54][55] introduce a method to account for that by incorporating contextual information at various tempo levels into a single distance measure. Other than that, there is a variety of methods that can handle tempo differences in the repeated parts [32, 39, 74, 54]. Equivalently to the homogeneity-based approaches, there are also techniques that apply an HMM state sequence representation to model a hierarchical description of the structure [69]. Aucouturier and Sandler [6] proposed an alternative. They compute a binary co-occurrence matrix based on the state sequence. This is inspired by the SDM. The matrix holds ones if two frames have the same state assignment, and zeros otherwise. The post-processing is similar to the already discussed ones in this section: smoothing kernel and stripe search.

Combined approaches Given multiple descriptors for segmentation, one may improve the performance of every single one by combining. Paulus and Klapuri [61][63] advise to relate descriptors and consequently compute a cost function based on the within-group dissimilarity, the amount of unexplained and the complexity. The within-group dissimilarity measures the probability of each pair of one group to be in that group; the amount of unexplained measures how much of the song is unstructured (not associated with a segment), and the complexity determines how many fragments are found. An optimal solution will tend to be less complex. Subsequently, the cost function is optimised in an unsupervised manner. All three aspects have a certain weighting, balancing them is crucial for the success of this method. In Figure 2.6 an example for that is displayed. The method proposed in Paulus and Klapuri [63] represents an example for a design that utilises all approaches mentioned in this chapter so far. First a set of candidate segments is created by a novelty-based method. Afterwards homogeneity- and repetition-based approaches are included in the cost function. The study comes to the conclusion that one main weakness of the presented method is that the success of the segmentation depends essentially on finding a set of possible segmentation boundary points. This bottleneck situation emphasises the importance of finding correct boundaries for music structure analysis.

2.1.3. Feature preparation for machine learning purposes

In the previous sections, we described how to assign a feature vector (e.g., chroma) to every point in time of a piece of music. As illustration, in Figure 2.7, we show an example of an audio wave and the corresponding spectrogram. At this point it is unclear how machine learning can be applied on this spectrogram matrix and

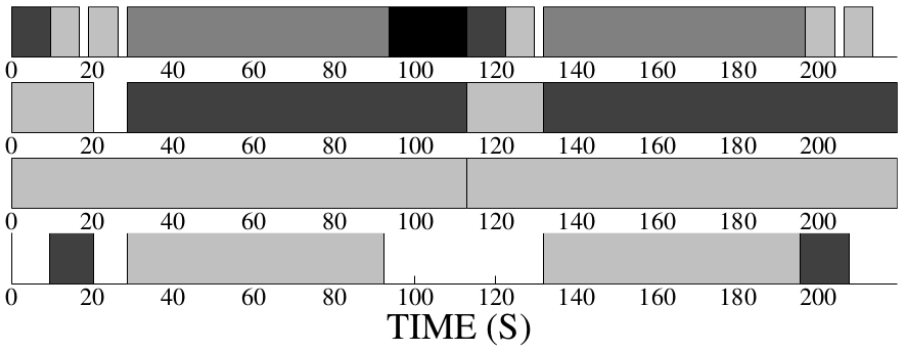

Figure 2.6: The effect of different weightings within the cost function on the final structure description. Top: Annotated ground truth. Second row: Analysis result with some reasonable values for the weights. Third row: Result with increased weight of the complexity term. Bottom: Result with a decreased weight for the term amount unexplained. Figure by Paulus and Klapuri [61].

how annotated boundary points can be used in order to learn from them. To feed the data to common machine learning algorithms and in particular neural networks, we require input sequences of equal size. For that purpose, we will assign equally sized spectrogram excerpts to each point in time. We indicated two of these neighbouring excerpts in the figure (purple). We will refer to an excerpt as one data instance $\hat{\mathbf{x}}$. This term is independent of the feature at hand. Finally, we also assign targets to all times. For this work we propose binary targeting. That means if a point in time is an annotated boundary point we set the target t to one, we assign the value zero otherwise. In Chapter 3 we will extend this method to account for the inaccuracy and rareness of boundary points in our data set.

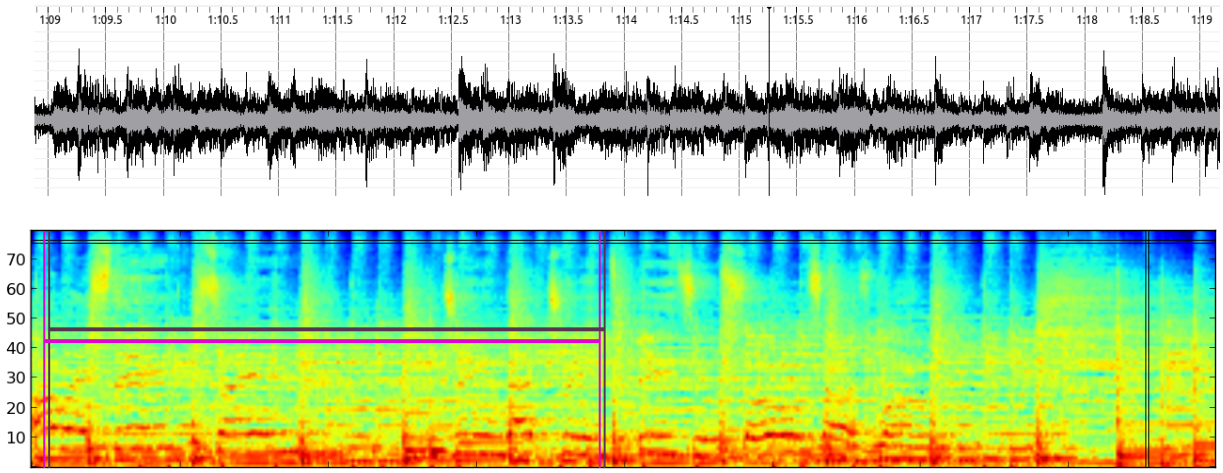

Figure 2.7: Visualisation of the Song “How beautiful you are” from the band The Cure. Top Panel: Visualisation of the sound-wave. Bottom Panel: The same signal mapped to the frequency space (mel spectrogram). Note that we indicated two neighbouring spectrum excerpt windows, each providing context for one particular time frame. Furthermore, we indicated one frequency bin (horizontal) and one time bin (vertical).

2.1.4. Data-driven models in MSA

To the best of our knowledge, there are only two studies in MSA that learn from data. In one of them a 832-dimensional hand-designed feature vector is computed for every time frame [83]. The vectors consist of spectral

parts, MFCCs, chroma, melody and rhythm features, as well as first and second derivatives of these components. As described in the previous section, the authors used a binary labelling to assign targets to corresponding feature vectors. Subsequently, they applied ada-boosting to find a classifier for their data. The second algorithm by [52] is currently scoring highest on the SALAMI data set. It also utilises mixed features to account for timbre (MFCCs), chroma and repetition. The authors apply an adaptation of Fisher's linear discriminant (FLD) as classifier. With respect to the data, both of the mentioned classifiers are not ideally suited for the problem for one reasons: None of them considers temporal context (other than calculating derivatives) and thus can not deal with temporal uncertainty of labels. There is, however, one type of model that is known to perform especially well facing label noise: neural networks. In MIR there are three studies that applied them successfully already: Schlüter and Böck [72] for onset detection, Boulanger-Lewandowski et al. [12] for chord recognition and Li et al. [47] for music genre classification.

2.2. Feed forward neural networks for machine learning

In the previous section we reviewed and discussed methods to extract structural information from music. We realised the importance of SSIMs and their post-processing for the extraction of segment boundaries in most MSA algorithms. For many approaches, well designed smoothing kernels for matrix correlation play a major role for the success of the particular method. Commonly, these kernels are hand-tuned. Thus, an optimal behaviour of those kernels cannot be guaranteed. It would be interesting to see if we could learn the kernels with the help of annotated sample data. For this purpose, we will apply a specific *adaptive basis model* (ABM), the *Feed-Forward Neural Networks* (FNNs) [58]. ABMs are models that map an input $\mathbf{x} \in \mathbb{R}^{D_x}$ to the corresponding output $\hat{\mathbf{y}} \in \mathbb{R}^{D_y}$ in the following manner

$$\mathbf{y}(\phi(\mathbf{x})) = \mathbf{p}_0 + \sum_{m=1}^{M} \mathbf{p}_m \vec{\Phi}_m(\vec{\phi}(\mathbf{x})) \tag{2.5}$$

$$= \mathbf{p}_0 + \sum_{m=1}^{M} \mathbf{p}_m \vec{\Phi}_m(\hat{\mathbf{x}}), \tag{2.6}$$

in which in our case, $\mathbf{x}$ denotes the raw audio input, $\phi(\cdot)$ denotes one or a combination of the features discussed in Section 2.1 and $\Phi_m(\cdot)$ the m'th *basis function* of the model. We will write $\phi(\mathbf{x})$ as $\hat{\mathbf{x}}$ from here on for brevity. Generally, the basis functions are parametric, thus we can write $\Phi_m(\hat{\mathbf{x}}) = \Phi_m(\hat{\mathbf{x}}; \mathbf{v}_m)$, where $\mathbf{v}_m$ are the parameters assigned to the basis function. We will use

$$\theta = (\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_M, \{\mathbf{v}_m\}_{m=1}^{M}) = (\mathbf{P}, \mathbf{V}) \tag{2.7}$$

to denote the set of all parameters in the model. We can learn θ from the data. Note that ABMs are non-linear models in general. Hence, it is in most cases not possible to derive an optimal setting for θ, thus we will only be able to compute locally optimal settings of θ rather than global ones. Nevertheless, a great variety of application showed that ABMs outperform linear models considerably [41, 82, 38, 16].

2.2.1. Definition

FNNs, also known as *multi-layer perceptrons* (MLPs), are built of a series of logistic regression models [11]. That means, that we first compute a linear combination $\mathbf{a}$ using one of the given data instances $\hat{\mathbf{x}}^n$ out of

the set of all data instances $\hat{\mathbf{X}} = \{\hat{\mathbf{x}}^n\}_{n=1}^N$ as input:

$$\mathbf{a}^{(1)}(\hat{\mathbf{x}}^n) = \mathbf{W}^{(1)} * \hat{\mathbf{x}}_n + \mathbf{w}_0^{(1)}, \tag{2.8}$$

in which $\mathbf{W}$ denotes the weights and $\mathbf{w}_0$ the bias. We will refer to this set of parameters as the first layer parameters of our network. The superscript (1) is used to denote this. Furthermore, we will call $\mathbf{a}$ the *activations*. In a second step we transform the activations via a non-linear differentiable function $h(\cdot)$ called the *activation function*:

$$\mathbf{z}^{(1)} = h^{(1)}(\mathbf{a}^{(1)}), \tag{2.9}$$

in which the activation function is an element-wise mapping. The quantity $\mathbf{z}$ are called the outputs of *hidden units* of the network. They will serve as input for the next layer. As we did before, we will linearly combine the hidden units

$$\mathbf{a}^{(2)} = \mathbf{W}^{(2)} * \mathbf{z}^{(1)} + \mathbf{w}_0^{(2)} \tag{2.10}$$

From here we can expand the process successively until our network reaches desired capacity and flexibility. The indexes of Equation 2.10 can be generalised to $l = 1, .., L$. Note that we can control the model capacity by determining the number of layers and by defining the size of each layer [3]. We will denote the last layer's output $\mathbf{z}^{(L)}$ as the *output of the network* $\mathbf{y}$. The activation functions are an important model choice for FFNs and need to correspond to the nature of the data and the assumed distribution of target variables. One could assign different activation functions to each layer, it is not common though. Frequently applied activation functions $h(\cdot)$ are the $\tanh(\cdot)$ and the sigmoid function $\sigma(\cdot)$ for binary classification problems and the soft max activation function for multiple-class problems. Note that the whole network is non-linear as soon as we have at least one non-linear activation function $h(\cdot)^{(l)}$

At this point we will briefly compare the general definition of an ABM (Equation (2.6)) to the definition of an FNN. Every hidden layer $\mathbf{z}^{(l)}$ forms an ABM, in which the weights $\mathbf{W}^{(l)}$ and $\mathbf{w}_0^{(l)}$ form the parameter set $\mathbf{P}$ and all parameters $\{\mathbf{W}^{(k)}, \mathbf{w}_0^{(k)}\}_{k=1}^{l-1}$ that were used to generate $\mathbf{z}^{(l)}$ form the set of parameters $\mathbf{V}$. And $\mathbf{z}^{(l)}$ itself is a basis function of the model $\mathbf{z}^{(l+1)}$. The distinction of $\mathbf{P}$ and $\mathbf{V}$ in θ is not much of use when discussing FNNs. This is why we will henceforth describe θ as

$$\theta = \{\mathbf{W}^{(l)}, \mathbf{w}_0^{(l)}\}_{l=1}^L \tag{2.11}$$

when speaking about a neural network. Note that the number of layers L corresponds to the number of adaptive weight matrices. In the next section, we will discuss how to adjust these parameters θ in a computationally effective way. We will close this section with a remark regarding the theoretical bounds of FNN models.

2.2.2. Network optimisation

In this section, we will explain how to estimate a locally optimal setting of θ with respect to network performance. While discussing this, we will realise that the process itself has parameters that need to be tuned. We will refer to them as the hyper-parameters φ. They may influence the performance of our networks in an essential way. For example, the network architecture, i.e., the number of layers L and the size of the activations,

[3] by choosing $\mathbf{W}$ in the desired size, thus $\dim(\mathbf{a}^{(q)}) \neq \dim(\mathbf{a}^{(l)})$ in general, $q, l \in \mathbb{N}_+, q \neq l$

are elements of φ. Often φ is hand tuned. In this work we will also present more automated alternatives. Both sets, the parameters θ and the hyper-parameters φ, will be learned from the data. For that purpose, we divide $\hat{\mathbf{X}}$ into three disjoint sets: the training set $\hat{\mathbf{X}}_{\text{train}}$ for optimising θ, the validation set $\hat{\mathbf{X}}_{\text{valid}}$ for optimising φ, and the test set $\hat{\mathbf{X}}_{\text{test}}$ for evaluating the network performance.

2.2.2.1. Optimising θ - Network training

Network training is a term that is used to describe how the set of parameters θ is adjusted by the given data, so that the network outputs $y(\hat{\mathbf{x}}_n) = y_n$ are in agreement with the targets $t(\hat{\mathbf{x}}_n) = t_n$ as much as possible. The agreement in that context is measured by the *objective function* E (also error or fitness function). How it is composed depends crucially on the problem at hand. We will discuss the most frequent problems and a natural choice of error function for every one of them in the next section following [11]. Following, we will show how the objective function is optimised interactively and in a computationally efficiently. To that end, we will introduce the backpropagation algorithm, a method that is most common to derive the gradient of the objective.

Objective function When given an arbitrary FFN architecture and differentiable activation functions, we can derive a closed form algorithm for a broad range of fitness functions if we can make the assumption that the error is decomposable into a sum of terms, one for each data instance at hand:

$$E(\theta) = \sum_{n=1}^{N} E_n(\theta). \tag{2.12}$$

With this formulation, we will be able to split the learning process using only small mini-batches of the entire data at a time. Initially, we will introduce the standard error functions for regression, logistic regression (binary-class problems) and multi-class regression problems. We will motivate them with intuitive assumptions about the relations of inputs $\hat{\mathbf{x}}_n$, the network outputs $y(\hat{\mathbf{x}}_n)$ and targets $t(\hat{\mathbf{x}}_n)$. Furthermore, we will see that all of the three presented functions fulfil the requirement of being decomposable per data instance.

Firstly, we consider regression where targets lie in $t_n \in \mathbb{R}$. Here, ideally, the ground-truth $t(\mathbf{x}_n)$ is Gaussian distributed having its mean in the network output $y(\mathbf{x}_n, \theta)$ and some precision β[4]. In this section, we will neglect the possibility of additionally considering conditional distributions as it is done for *Bayesian Neural Networks* (BNNs)[5]. Thus, assuming N independent, identically distributed data instances we can derive the corresponding likelihood function

$$p(\mathbf{t}|\hat{\mathbf{X}}, \theta, \beta) = \prod_{n=1}^{N} p(t_n|\hat{\mathbf{x}}_n, \theta). \tag{2.13}$$

Note that, it does not rely on the assumptions regarding the regression problem, hence we will reuse the formulation later again. Consequently, we take the negative logarithm in order to derive a decomposable minimisation problem. Furthermore, we insert a Gaussian distribution [6] for the target distribution $p(t_n|\hat{\mathbf{x}}_n, \theta, \beta)$ with mean in y_n and precision β as claimed before:

$$\ln p(\mathbf{t}|\hat{\mathbf{X}}, \theta, \beta) = \underbrace{\frac{\beta}{2} \sum_{n=1}^{N} \{y(\hat{\mathbf{x}}_n, \theta) - t_n\}^2}_{E_D} - \frac{N}{2} \ln(\beta) + \frac{N}{2} \ln(2\pi). \tag{2.14}$$

[4] The precision describes the inverse variance $\beta = 1/\sigma^2$

[5] not to be confused with Bayesian Networks

[6] $\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{(D/2)}} \frac{1}{|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)\right]$, in which D denotes the dimensionality and $|.|$ the determinant.

We recognise E_D to be the *sum-of-squared-errors*. Because the other two terms do not depend on the data we can neglect them during training.
Secondly, we will be concerned with error functions for binary classification, $t_n \in \{0, 1\}$. We will assume that we can interpret $y(\hat{\mathbf{x}}_n, \theta)$ as a probability , thus $y(\hat{\mathbf{x}}_n, \theta) \in [0, 1]$. For the underlying network architecture, that means that the last layer's activation function is commonly a sigmoid, such a unit is called a *logistic unit*. Consequently, we assume $p(t_n|\hat{\mathbf{x}}_n, \theta)$ to be Bernoulli distributed.

$$p(t_n|\hat{\mathbf{x}}_n, \theta) = y(\hat{\mathbf{x}}_n, \theta)_n^t \{1 - y(\hat{\mathbf{x}}_n, \theta)\}^{1-t_n} \tag{2.15}$$

Again we take the negative log-likelihood under the assumption that (2.13) holds.

$$E(\theta) = \ln p(\mathbf{t}|\hat{\mathbf{X}}, \theta) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \tag{2.16}$$

We will call error function the *binary cross-entropy*. It will be particularly important for this work.
Finally, when given multiple classes we will use the 1-of-K folding strategy to encode targets, $t_k = \{0, 1\}$, $k = 1, ..., K$. Here, each input is assigned to one of K classes that are mutually exclusive. Thus $|\vec{t}| = 1$. Equivalently to the binary problem, $y(\hat{\mathbf{x}}_n, \theta)_k \in [0, 1]$ and the last layer for multi-class regression consists of a softmax function to guarantee $|\mathbf{y}| = 1$. For these reasons, we can describe $p(\vec{t}_n|\hat{\mathbf{x}}_n, \theta)$ as follows:

$$p(\vec{t}_n|\hat{\mathbf{x}}, \theta) = \prod_{k=1}^{K} y_{n,k}(\hat{\mathbf{x}}, \theta)^{t_{n,k}}. \tag{2.17}$$

Again we take the negative log-likelihood over all data instances under the assumption that (2.13) holds.

$$E(\theta) = \ln p(\mathbf{T}|\hat{\mathbf{X}}, \theta) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{n,k} \ln y_k(\hat{\mathbf{x}}_{n,k}, \theta) \tag{2.18}$$

To sum up, we found natural choices to fit output unit activation function and matching error function for three standard problems. For regression we utilise linear outputs and a sum-of-squares error, for binary classifications we apply logistic outputs and the binary cross-entropy error function, and finally, for multi-class classification, we employ softmax outputs with the multi-class cross-entropy error function. In network optimisation, it became common use the negative log-likelihood function as objective. In that fashion, we have a minimization problem because $E < 0$. This is however only a notation convention.

Objective optimisation In order to find local minima in the weight space we are looking for

$$\vec{\nabla}_\theta E \stackrel{!}{=} 0. \tag{2.19}$$

Thus, the aim is find stationary points of $E(\cdot)$. One frequently applied way is to initialise θ with some θ_0 and then successively computing

$$\theta^{(\tau+1)} = \theta^{(\tau)} + \triangle\theta^{(\tau)} \tag{2.20}$$

in which τ denotes the iteration step and $\triangle\theta^{(\tau)}$ labels the weight vector update. We hope to have chosen the mapping $\triangle$ in that way that $E(\theta^{(\tau)}) \to E_{\text{min}}$ for $\tau \to \infty$, in which E_{min} is a minimum. Many algorithms use gradient information to compute $\triangle\theta^{(\tau)}$. Thus, for these methods it is required to compute $\vec{\nabla}_\theta E(\theta)$. Specifically, the gradient descent optimisation computes

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \eta\vec{\nabla}_\theta E(\theta)). \tag{2.21}$$

were η is known as the *learning rate*. At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function, and so this approach is known as *gradient descent* or *steepest descent*. Computationally, the algorithm covers two stages. In the first stage, the derivatives of the negative log likelihood or error function with respect to the parameter set must be evaluated. In the second stage, the derivatives are then taken to compute the adjustments to be made to the weights. Rumelhart et al. [70] introduced the most common technique today. In its original formulation it is a minimisation method however can be turned into a maximiser by flipping the sign (gradient ascent). It is important to note that the two stages are distinct and thus independent. Notice that in this formulation the gradient of the objective is computed with respect to the entire training set. Thus, each update requires the entire training set to be taken into account. Techniques that use the full batch of data to compute the gradient are called *full batch methods*. For full batch optimisation, there exist more efficient methods, e.g., conjugate gradients and quasi-Newton methods, that are more robust and faster than simple gradient descent [59, 22, 27]. These algorithms are guaranteed to converge, that means that the error function always decreases or stagnates at each iteration. However, today's NNs are usually trained with only a subset of the normally large data set at hand per iteration [44]. Those *on-line methods* rely on the assumption that we can split the objective with respect to terms of every single data point (see Equation (2.12)). On-line gradient descent, also known as *sequential gradient descent* or *stochastic gradient descent*, makes updates of the parameter set with exactly one single data point x_n.

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \eta \nabla E_n(\theta^{(\tau)}) \tag{2.22}$$

More generally, *mini-batch algorithms* use a subset of the data for each iteration. The updates are successively randomly repeated. To illustrate the advantage mini-batch methods have in comparison to full batch methods, assume we have a data set A. To increase its size we clone each data point and call the set B. Thus we have an enormous amount of redundant data in B. When using a full-batch approach to optimise the corresponding objective on set B we need twice the time as for set A, although analytically this action only multiplies the error function by a constant factor of 2. Thus, there is no gain for the model in using B, it is equivalent to using the original error function. On the other hand, mini-batch methods will stay unaffected by this action. Moreover, mini-batch methods escape from local optima more likely than the full batch ones. More specifically, the gradient derived with the entire data set will point to the closest local minimum. Contrary to that, the gradient derived with a data sub set will not necessarily point in the same direction because this subset may statistically differ from the entire set. Note that, due to the randomness of the update (see Equation 2.22), we will achieve different results for every run of this algorithm. To compare the quality of each run we keep a subset of the data on which we do not train but evaluate our runs on. One run can be seen as the outcome of a random experiment in which the random variable is the value of the evaluation.

Backpropagation Henceforth, we will describe a method to efficiently compute the gradient of an error function given an FNN architecture, the so-called *backpropagation algorithm*. Initially, we consider the gradient of the error function with respect to θ. For objectives for which the condition from Equation 2.12 is valid, we derive

$$\bigtriangledown_\theta E(\theta) = \sum_{n=1}^{N} \frac{\partial E_n(\theta)}{\partial \theta}. \tag{2.23}$$

Subsequently, we consider how to compute $\bigtriangledown_\theta E_n(\theta)$ For that purpose, we apply the chain-rule to the gradient with respect to a particular set of weights from a layer l.

$$\frac{\partial E_n(\theta)}{\partial \mathbf{W}^{(l)}} = \frac{\partial E_n(\theta)}{\partial \mathbf{z}^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{a}^{(L-1)}} \cdots \frac{\partial \mathbf{a}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \frac{\partial \mathbf{a}^{(l)}}{\mathbf{W}^{(l)}} \tag{2.24}$$

$$= \frac{\partial E_n(\theta)}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{a}^{(2)}} \cdots \frac{\partial \mathbf{a}^{(L-q)}}{\partial \mathbf{a}^{(L+1-q)}} \frac{\partial \mathbf{a}^{(L+1-q)}}{\mathbf{W}^{(L+1-q)}} \tag{2.25}$$

in which $q = L - l + 1$ according to the definition of an FFN. Note that, the l-notation labels the forward path trough the network whereas the q notation labels the backward path. We specify $\frac{\partial E_n(\theta)}{\partial \mathbf{z}^{(L)}} = \frac{\partial E_n(\theta)}{\partial \mathbf{y}}$ as $\delta^{(0)}$. Specifically, for the sum of squared errors $\delta^{(0)} = y - t$ and for the cross-entropy error $\delta^{(0)} = \frac{t}{y} + \frac{1-t}{1-y}$. Next, we define the error according to the layer depth $\delta^{(1)}, ..., \delta^{(L)}$ to be

$$\delta^{(q)} \equiv \frac{\partial E_n(\theta)}{\partial \mathbf{a}^{(q)}} \tag{2.26}$$

This definition simplifies the gradient computations. Consider the error at $q + 1$:

$$\begin{aligned} \delta^{q+1} &= \frac{\partial E_n(\theta)}{\partial \mathbf{a}^{(q)}} \frac{\partial \mathbf{a}^{(q)}}{\partial \mathbf{a}^{(q+1)}} \\ &= \delta^{(q)} \frac{\partial}{\partial \mathbf{a}^{(q+1)}} \left(\mathbf{W}^{(q+1)} * h(\mathbf{a}^{(q+1)}) + \mathbf{w}_0^{(q+1)} \right) \\ &= \delta^{(q)} \mathbf{W}^{(q+1)} h'(\mathbf{a}^{(q+1)}), \end{aligned} \tag{2.27}$$

where we applied the previous definition and Equation (2.9) and (2.10). We can successively compute the gradient by iteratively computing $\delta_0, \delta_1, ..$ up to the layer q, that contains the weight we want to update. According to Equation 2.25, the weight gradient simply is computed as follows:

$$\frac{\partial E_n(\theta)}{\partial \mathbf{W}^{(q)}} = \delta^{(q-1)} \frac{\partial \mathbf{a}^{(q)}}{\partial \mathbf{W}^{(q)}}. \tag{2.28}$$

The advantage of this procedure becomes more clear when considering the computation order. First, we feed a new data input x_n into the network and compute the output y_n with the current weights. Afterwards, we evaluate $\delta^{(0)}$ with y_n and compute the weight gradient corresponding to the last layer. Next, we compute $\delta^{(1)}$ and the corresponding weight gradient. We continue with this strategy until we reach the deepest layer of the network. Note that, to compute $\delta^{(q+1)}$ we need $\delta^{(q)}$ and that we can simply request from the memory instead of computing it again. Because the process starts computing in the last layer, it is called error backpropagation.

Additional aspects to objective optimisation In this section, we have seen how to compute gradient information with respect to every layer's weights. With this information we can update the weights of our network with the update rule Equation 2.22 starting with the output layer and proceeding with deeper layers. There are a few parameters in this process that may influence its performance enormously. We discuss them in the following:

(1) The batch size. If we take the full batch approach than we may more likely end up in a local optimum whereas when we take a very small batch size the gradient may overshoot close to the optimum that is desired to achieve. It is important to note that the former is a much bigger threat to the method than the latter.

(2) The learning rate, as mentioned in Equation (2.22) may be a function of τ, too. For simplicity reasons, it is often a exponentially decreasing function of the form

$$\eta^{(\tau+1)} = b_\eta \eta^{(\tau)}, \tag{2.29}$$

in which b_η labels the learning rate decay [44]. The motivation for this is to take bigger steps in the beginning of the procedure to allow space exploration and to take smaller steps towards the end to achieve convergence. Another idea is to apply adaptive step sizes. For example the learning rate could be decreased when the training error increases and vice versa.

(3) The momentum [58]. We extend Equation 2.22 by a "memory" term to avoid zig-zag behaviour. [7] In order to do so we introduce the momentum:

$$m_k = \theta_k - \theta_{k-1}. \tag{2.30}$$

Thus, the weight update (Equation 2.22) becomes:

$$\theta_{k+1} = \theta_k - \eta_k \bigtriangledown_k E(\theta_k) + \mu_k m_k \tag{2.31}$$

where $0 \leq \mu_k \leq 1$ controls the importance of the momentum term. The discussion we had about b_η applies equivalently to μ_k.

Usually, one starts with a high learning rate and a low momentum. During training, one would gradually decrease the learning rate and increase the momentum. Where the idea is to allow more exploration at the beginning of the learning and force convergence at the end of learning. With the introduction and adjustments of these new parameters we try to minimise the drawbacks of gradient descent based methods: slow convergence close to the minimum or the possibility of exhibiting increasing "zigzag" behaviour when the gradients point nearly orthogonally to a close minimum point. However, the weight space we try to optimise in is huge and very often it is already satisfying to find some local minimum of the error.

2.2.2.2. Optimising φ

Apart from the set of hyper-parameters that can be adjusted while network training there will remain some hyper-parameters that cannot. For one, there are the hyper-parameters related to the gradient descent algorithm such as the learning rate and decay, the momentum and momentum rise, the batchsize and the number of epochs (iterations over the training set) the network needs to be trained. Furthermore, there are hyper-parameters that have no real or integer value assigned to them such as the network topology and there hyper-parameters regarding data sampling (which we will explain later in detail). Commonly, those hyper-parameters are hand-tuned, which requires an enormous amount of expert knowledge about NNs. Alternative approaches exist though. In particular, for real-valued evaluators (i.e., the objective in our case) there exist approaches for hyper-parameter optimisation. We will present one in this section. The main reason why these approaches are not further explored yet could be that evaluations are quite time intensive and we do not aim for global optimisation of the NN anyway because the weight space is too enormous [11].

[7] An alternative way to minimise "zig-zagging" is to use the method of conjugate gradients (see e.g., (Nocedal and Wright [59], ch 5) or (Golub and Van Loan [29], Section 10.2)).

Evolutionary strategies: CMA-ES In this section, we will assume the reader to be familiar with the basic concepts of *Evolutionary Strategies* (ES) as subdivision of *Evolutionary Computing* (EC). Note that, it needs to be distinguished from Genetic Algorithms that are often identified as EC which is not true in general. ES are favoured for real-valued vector optimisation within the field of EC [20].

As described in [20], the key idea in ES is to evolve a population by noise drawn from a multivariate Normal distribution (a.k.a. mutation). The parameters defining the particular Normal distribution are thereby carried by the *individual* (i.e., in our case a specific network hyper-parameters configuration) itself as part of its *genotype*. It is important to remark that the parent selection simply applies a uniform distribution. It is thus unbiased. The recombination and survivor selection operators on the other hand are comparable to other methods and do not need any further notes.

Today's state of the art algorithm is called *Covariance Matrix Adaptation* (CMA or CMA-ES). Here the mutation is done with the help of a full covariance matrix in order to adjust the noise to a given energy landscape (corresponding to a minimisation problem). In the following, we will illustrate the algorithm in a nutshell rather than introducing the motivation for certain steps. If interested the reader may refer to Hansen [36]. Before we proceed, however, we want to note that in our case, the objective function for the CMA will be an evaluator for the trained NN. Training an NN may be time intensive. Furthermore, an NN can be seen as a random variable so we will need to average over a few runs to evaluate one individual. Thus we aim to evaluate as few population members (i.e., one hyper-parameter settings) as possible.

First of all, we fix the population size λ. It is necessarily larger than 2 but generally larger than 4. This hyper-parameter is one of the most important decisions when dealing with a time consuming fitness evaluation. On the one hand, we want a high number of samples the Normal distribution can be adjusted with. On the other hand, we want to keep the time the algorithm is consuming as low as possible. Secondly, we initialise the the Normal noise parameters such as the mutation step size σ, the mean value $\mathbf{m}$ and the covariance matrix $\mathbf{C}$ of the search distribution. Moreover, we set the evolution path of σ, p_σ and $\mathbf{C}$, $p_\mathbf{C}$ to be zero vectors. Consequently, we start the evolution process. It is terminated with respect to certain constraints such as a number of maximal iterations is reached or an optimisation goal is achieved. Henceforth, until the termination criteria are met, the algorithm is creating λ individual samples from the Multivariate Gaussian $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C})$. These samples are consequently evaluated by the fitness function, in our case some network evaluator, and sorted by their results. Influenced by these results the parameters $\mathbf{m}$, p_σ, $p_\mathbf{C}$, $\mathbf{C}$ and σ are updated in this order. For further information on the update rules please review Hansen [36].

We will apply the described algorithm mainly for learning rate and mutation rate related matters, thus the search space will be relatively low-dimensional. Another method currently exploited by Snoek et al. [79] is to apply Gaussian processes to optimise the search. The method's idea is somewhat similar. This is why we will not experiment with it. Some hyper-parameters will not be optimisable within these frameworks. The architecture, for example, is a complicated hyper-parameter including the number of layers and their sizes. We can imagine to tune this by Genetic Programming. Although it is known that the corresponding trees are as complex as one allows them to be, this algorithm offers high flexibility in building its individuals[20].

2.2.3. Network regularisation

The enormous complexity of a neural network model can easily lead to model *over-fitting*. That means that the model fits the specific data rather than the underlying relationship . In order to still allow large architectural

complexity, we can restrict the capacity of an NN model with means of prior assumptions about the underlying structure of our data.

In order to achieve invariance with respect to linear transformations, one option is the application of Gaussian priors for the network weights (for more detail see [11], Section 5.5.1). Another approach is concerned with avoiding over-fitting by stopping the training process early, the so-called *early-stopping*. In doing so, we check the validation error until it rises and stop training at this point. The most important network regularisers, however, are the regularisers that ensure that predictions stay unchanged under transformations of the input variables (for more detail see [11], Section 5.5.2). In object recognition for images, transformation invariance may refer to scaling or translation invariance. We can apply the same concept to musical features. Thus, when classifying a piece, the prediction should not be dependent on say the quality of the recording or be sensitive to small rhythm changes. But transformations of this kind do produce significant changes in the raw data. If given large enough data sets an FNN can learn to sufficiently approximate the invariance. In the literature, we distinguish between four main approaches to help the network learn the invariances:

(1) Firstly, the training set can be enlarged by adding replicas of the training set to itself while these replicas must be transformed according to the invariances. In image recognition, for example, we would rescale the image or change the position of the object we aim to recognise. Although this approach is easily implemented and can achieve great generalisation success [75], it is also computationally costly. For this method to work, it is crucial to know the transformations we want to make our model invariant against. Finding the entire set of possible transformations is the bottle neck of this technique.

(2) Secondly, there is the so-called *tangent propagation*. Here, a regularisation term is added to the objective function which penalises changes in the model output when the input is transformed. Unfortunately, we suffer from the same problem as we did in approach (1) because in order to apply this approach we need to know a set of transformations.

(3) Thirdly, we can tackle the problem at an earlier stage and create features that are invariant under certain transformations. Any classifier that is using those will necessarily respect the invariances. An example for that are chroma vectors that are invariant again transpositions by full octave. However, as in the latter two cases, this approach is rarely applicable to music because it may be difficult to find hand-crafted features with the required invariances that do not also discard information that can be useful for discrimination.

(4) Finally, there is a method that does not require to specify the exact transformations. It rather learns them. More specifically, we build the ability to learn certain invariances into the model. A popular option are convolutional neural networks, which will discussed extensively next.

2.2.3.1. Convolutional layers

In music analysis, any of the features discussed in Section. 2.1.1 will map to the (possibly distorted) time-frequency space. It is intuitive to assume that we can identify local patterns that are globally applicable and thus can be helpful for classification. For example the network could learn to identify triads on its own. We will refer to this aspect as *spatial relationships*. This assumption can help to decrease the model capacity significantly. For this purpose we introduce *convolutional layers*. They build on three major mechanisms: (1) local receptive fields, (2) weight sharing, and (3) sub-sampling. In a convolutional layer, units are mapped to planes called *feature maps*. To create a feature map the network only takes small subregions from the lower layer (original

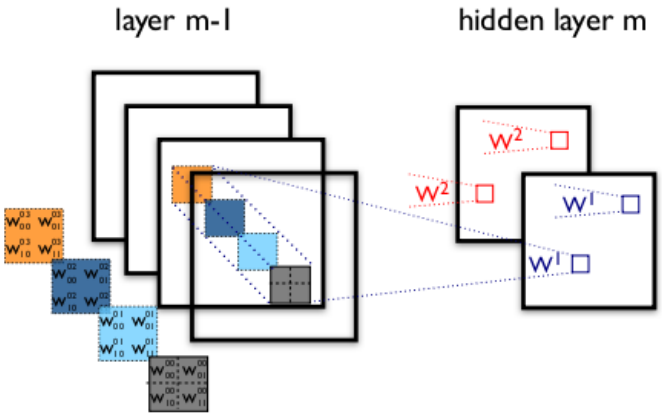

Figure 2.8: This is a symbolic excerpt from a network with convolutional layers. Illustrated are 4 feature maps of layer $(m-1)$ and 2 feature maps (h^0 and h^1) of layer m. Shared weights are indicated by same color. Each weight in layer m is learned by a patch of outputs from layer $m-1$. Figure by LISA-lab [48]

input or other hidden layers) as input. All units of one feature map are constrained to share weights. Let us illustrate the concept by one example: say a feature map consists of 80 units arranged in a 8×10 grid, with a single unit taking inputs from a 4×5 pixel patch of the original input that could be an image. The entire feature map thus has ($4 \cdot 5$=) 20 weight parameters and one bias parameter. For an illustration of the process see Figure 2.8. After the convolution stage (realisation of concepts (i) and (ii)), the inputs are processed according to Equation (2.8). In the next stage, the outputs of the convolutional layer (feature maps) are taken as inputs for the sub-sampling layer. This step is optional but often useful for dimensionality reduction. The sub-sampling layer applies a certain sub-sampling function to a number of locally connected units in the feature map. For instance, extending the example, imagine we have 8×10 sized feature maps and 2×2 sub-sampling regions. This means, the feature map is compressed to 4×5, reducing the number of inputs by the factor 4 from 80 to 20 units. Typical sub-sampling functions in this context built the mean or the maximum. The receptive fields of the sub-sampling layers are chosen to be contiguous and non-overlapping. Thus in our example there would be half the number of rows and columns in the feature maps after sub-sampling. Note that the term "convolutional layer" comes from the analogy to convolving the input signal of a unit with a "kernel", where the kernel parameters are the shared weighs learned by the network. Thus it is comparable to methods described in Section 2.1.2. But in contrast to hand-designed kernels, these kernels are adjusted by the data. The process of learning kernels can be seen as local feature extraction [43]. Convolutional layers are therefore sometimes referred to as feature extraction unit, whereas layers with no weight constraints, so-called *fully connected layers*, are the classification unit of a network [65]. We find this priciple displayed in Figure 2.9. Following this idea, we mostly need to detect multiple features to build a sufficiently accurate model, hence in general there will be multiple feature maps in the convolutional layer, with their own set of weights and bias parameters. This process of feature development and classification is refereed to as *deep learning*. Its success relies on the increase of invariance per layer. [42] achieved the first great successes with CNNs on handwritten digit recognition. These studies followed numerous others. Note that backpropagation based stochastic gradient decent is still applicable to this modified layer. The last remark is concerned with the dimensionality of the convolution. It is straight forward to extend the 2-dimensional convolution to the 3-dimensional case [40].

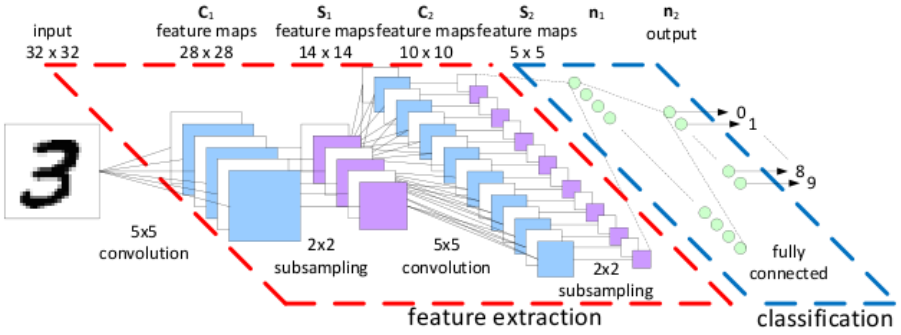

Figure 2.9: A typical network with convolutional architecture. Deeper layers are convolutional with optional pooling stage, follwed by a number of fully connected layers. The input to each layer are all features maps of the previous layer. Figure by Peemen et al. [65]

2.2.3.2. Dropout

Dropout is a form of network regularisation [80]. It can be described as a statistical method that uses multiple models and averages over them in each iteration of the training process. More specifically, consider an FNN with L hidden layers. Recall that $\mathbf{a}^{(l)}$ are the activations and $\mathbf{z}^{(l)}$ are the outputs of layer l, $l \in \{1, 2, ..., L\}$. We call $\mathbf{z}^{(l-1)}$ the input for layer l according to the depth of the layer. Now , we compute a random vector $\mathbf{r}^{(l)}$ that is Bernoulli distributed. Consequently, we multiply $\mathbf{r}^{(l)}$ element-wise[8] with $\mathbf{z}^{(l)}$

$$\tilde{\mathbf{z}}^{(l)} = \mathbf{r}^{(l)} \odot \mathbf{z}^{(l)}, \tag{2.32}$$

i.e., with a certain probability $(1 - p)$ the output is omitted from the network training. We call $\tilde{\mathbf{z}}^{(l)}$ the *thinned output*, which is subsequently used as input for layer $l + 1$:

$$\mathbf{z}^{(l+1)} = \mathbf{W}^{(l+1)}\tilde{\mathbf{z}}^{(l)} + \mathbf{w}_0^{(l+1)}. \tag{2.33}$$

This procedure can be repeated in any layer we desire. Note that we can train this network with backpropagation supported gradient descent as discussed in 2.2.2. The dropout process is thereby repeated separately for each mini-batch. For predicting test outputs we will obviously not use thinned outputs any more but rather utilise the complete network. The reason for the good generalisation of a network when applying dropout is that dropping certain connections causes the network to train towards the model mean of the set of all possible models. Whereas this statistical averaging is not given when using all connections. The method is especially appealing because this generalisation performance. This also effects the sensitivity to iteration related over-fitting, i.e., it gets less important at which iteration we stop training. For brevity we will henceforth use the term *dropout layer* to describe fully connected layer trained with the dropout method.

2.2.4. Visualisation of neural networks and inference

In data analysis, the complexity of models and the related danger of over-fitting are well known. One of the first studies that is concerned with visualising how well an FNN model's capacity is used by the training data is Zeiler and Fergus [87]. Additionally, we will discuss the work of Simonyan et al. [76] who were mapping the importance of pixels of the input signals to the classification in order to achieve a saliency map. These

[8] $\mathbf{a} \odot \mathbf{b} = \text{diag}\{\mathbf{a} \cdot \mathbf{b}^T\}$

techniques and their development could be useful for making informed decisions regarding the network architecture. For the purpose of giving "insight into the function of intermediate feature layers and the operation of classifiers", Zeiler and Fergus [87] apply the so-called *Deconvolutional Networks* (deconv net) [88] to diagnose common CNNs. In particular, each layer of the a CNN has a counter part that recovers the layers' action. In that way, every feature is mapped to pixels from the corresponding layer input. For every pooling layer, there is an unpooling layer. Since pooling is a non-invertible function of the inputs, the deconv-net needs to save switch variables to remember the location of the maximum taken by the pooling function, so subsequently the deconv net can place the reconstructed layer inputs appropriately. Accordingly, the so-called *rectification unit* of a deconv layer undoes the non-linearity. Afterwards, the filter unit transposes the learned filters (kernels) and applies them to output of the deconv rectification unit. The approach is illustrated in Figure 2.10 . The most

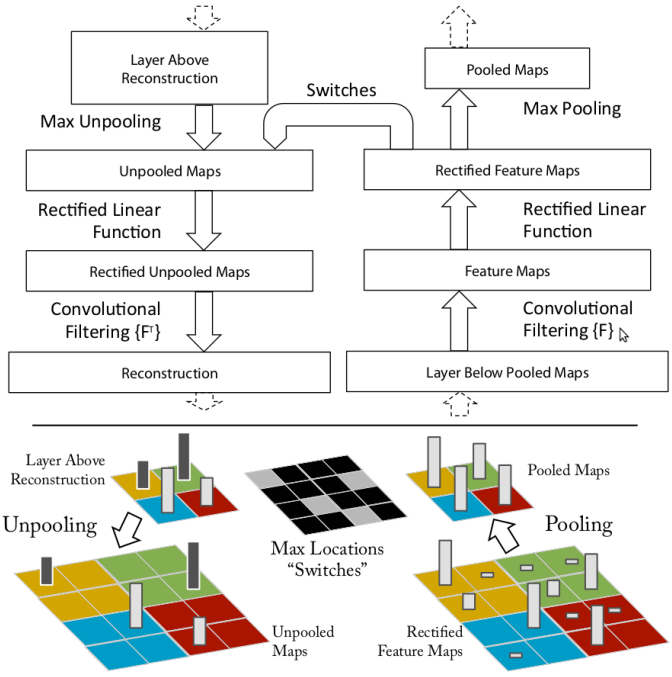

Figure 2.10: Illustration of the visualisation principle in Zeiler et al. [88].

important ability of this construction for image recognition is that the authors are able to interpret the role of every single layer. They found that lower layers detect edges whereas higher layers are more abstract. For example, one feature map detects grass in the background of a picture. Additionally they found that lower layers develop in a few epochs whereas higher ones need considerably longer. Unfortunately, this kind of insight is much harder to achieve for musical content because we cannot infer directly from our features. More important for music segmentation is the insight about how much the model capacity is leveraged. For example, when we can only find extremely low or high frequencies in the set of filters, we may need to shrink the size of the filters or adjust the filters. A generalisation of the deconv reconstruction procedure is provided by Simonyan et al. [76]. The authors compute so-called class *saliency maps* for each class and image combination. In this manner, they can show where the pixels are located that lead to the specific class decision. The method is gradient-based. Specifically, we approximate the activations of the very last layer $\mathbf{a}^{(L)}$ by the first-order Taylor expansion

$$\mathbf{a}^{(L)} \approx \frac{\partial \mathbf{a}^{(L)}}{\partial I}\bigg|_{I_0}^{T} I + \mathbf{b}_0 = \mathbf{b}^T I + \mathbf{b}_0 \tag{2.34}$$

in which I is the input of the network (not the layer) and $\mathbf{b}_0$ some bias. $\mathbf{b}$ is the derivative of $\mathbf{a}^{(L)}$ with respect to $\mathbf{b}$ at the point (network input) I_0. We can derive $\mathbf{b}$ by backpropagating the network input (not the weight parameter) through the net. $\mathbf{b}$ can also be interpreted as a map highlighting those pixels that need to be changed the least to affect the activations (network output) the most. In image classification, one would hope that these pixels correspond to the object rather than the ambient. In spectrograms we may be able to see which time context and which frequency bands are mostly used by the network . In order to obtain the saliency map M_{ij} for a certain picture and class we compute

$$M_{ij} = |\mathbf{b}_{h(i,j)}| \tag{2.35}$$

in which $h(i, j)$ is the index of the element of w, corresponding to the input pixel in the i-th row and j-th column. One example of a class saliency map is shown in Figure 2.11.

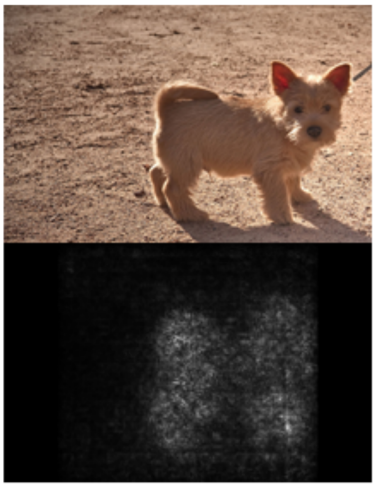

Figure 2.11: One saliency map for one picture and one class [76]. Top: Original picture Bottom: Saliency map. In this example, the light pixels that do define the class affiliation correspond to the object(the dog). A result expected for object recognition tasks.

2.3. Evaluation

2.3.1. F-measure

Due to the necessary temporal context, the evaluation of predictions may be difficult, and in particular can not be reflected by the error (derived by the objective) of the network. We illustrate this argument in Figure 2.12. On the panel, we show the network output in green, the ground truth in blue and a set of possible boundary predictions in red. At this point, the reader should not be concerned with how we computed the curve or the prediction. For that concern please see Section 3.5. The first phenomenon we observe is that the predictions may not exactly be on the ground-truth labels and that there may be multiple labels that correlate to the same ground-truth label, additionally there may be predictions that do not refer to any ground-truth as well as vice versa. A well designed measure will take these aspects into account. It seems intuitive to define a parameter that defines how wide the window is in which we assume the prediction to agree with the ground-truth. We will refer to this variable as the *time tolerance* τ.

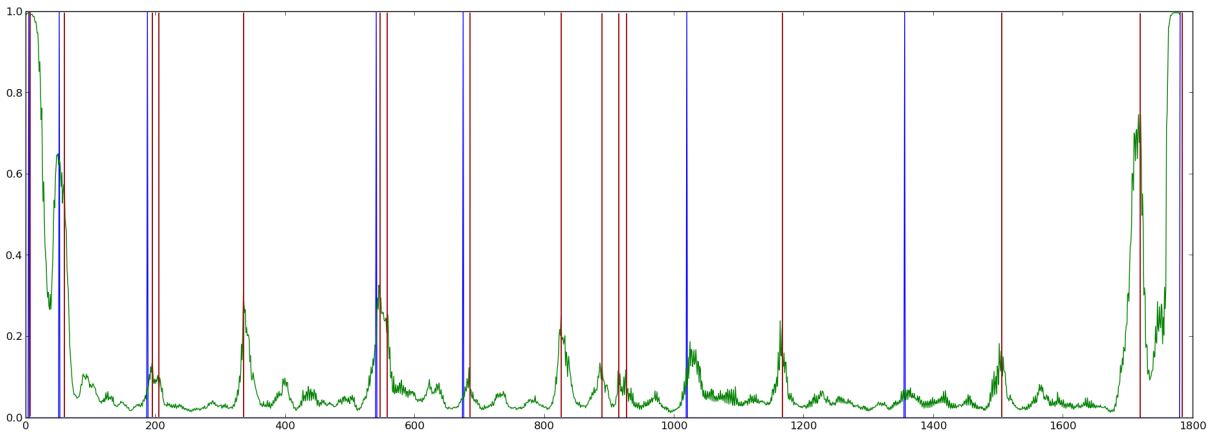

Figure 2.12: For every point in time, the network computes an output. This output can be interpreted as the probability for this point to be a boundary point. Concatenating all points, we obtain a curve for the entire musical piece (green). Ordinates of that curve that have the highest probability value in a local environment will be predicted boundary points. Derived predictions (red) can be compared to the annotated ground-truth (blue).

To evaluate the mutual agreement of a set of segmentation boundaries, the most frequently applied measures are the recall-rate, precision-rate and F-measure [45]. These necessarily need to account for some temporal inaccuracy (deviation) to be allowed without penalty, i.e., the time tolerance. In that context, the set of annotated labels is denoted as $\mathbb{F}_A$ and the set of predictions $\mathbb{F}_P$. The *precision-rate* R_P and the *recall-rate* R_R compare the elements of the two sets in a pairwise manner.

$$R_P = |\mathbb{F}_A \cap \mathbb{F}_P| / |\mathbb{F}_P| \tag{2.36}$$

$$R_R = |\mathbb{F}_A \cap \mathbb{F}_P| / |\mathbb{F}_A| \tag{2.37}$$

These two rates subsequently define the *F-measure* (F-score)

$$F = \frac{2R_R R_P}{R_R + R_P}. \tag{2.38}$$

Alternatively one can compute the average time between an annotated (ground-truth) label and a predicted one [83]. Also closely related to the F-measure is the *rand index* proposed by Hubert and Arabie [37]. All of these measures, however, have in common that they make pairwise comparisons. This leads to the weakness that none of the measures can evaluate the hierarchy or order of annotated segments. For instance, if a part has higher and lower level segment boundaries no measure penalises not found lower boundaries less than hierarchically higher valued boundaries. In other words, a multi-class segmentation evaluation is needed. First approaches for that are provided by Chai [15] and Paulus and Klapuri [61]. However, since we ignore segment hierarchies in this work this problem is not of interest yet.
In this study we will rely on the F-measure F_τ with $\tau = \pm 3s$ and $\tau = \pm 0.5s$. This refers to the standard of Music Information and Retrieval Evaluation eXchange (MIREX) 2009. MIREX is a framework for evaluating music information retrieval algorithms [2]. Results published in the context of this campaign will serve as benchmarks for this study.

2.3.2. Data set

We evaluate our algorithm on a subset of the Structural Analysis of Large Amounts of Music Information (SALAMI) database [78]. 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 [4]. A part of the dataset was also used in the "Audio Structural Segmentation" task of the annual MIREX evaluation campaign in 2012 and 2013 [3]. Along with quantitative evaluation results, the organisers published the ground truth and predictions of 17 different algorithms for each recording. By matching the ground truth to the public SALAMI annotations, we were able to identify 487 recordings. These serve as a test set to evaluate our algorithm against the 17 MIREX submissions. 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.

3. Method

In this chapter, it is described how our classifiers, i.e., the neural networks and their post-processing, were designed.

3.1. Feature extraction

For each audio file, we compute a magnitude spectrogram with a window size of 46 ms (2048 samples at a rate 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). We follow [72] in normalising each frequency band to zero mean and unit variance, so that we can bring the input values to a range suitable for neural networks. Finally, to allow the CNN to process larger temporal contexts while keeping the input size reasonable, we sub-sample the spectrogram by taking the maximum over three, six or twelve adjacent time frames (without overlap), resulting in a frame rate of 14.35 fps, 7.18 fps or 3.59 fps, respectively. We will refer to these frame rates as `high`, `std` and `low` resolutions. Beyond that, we tried training on MFCCs, chroma vectors, fluctuation patterns and self-similarity matrices derived from those, but mel spectrograms performed best.

3.2. Neural network architecture

For all further experiments, we set a base architecture that was determined by preliminary experimentation. We will vary this architecture gradually for optimisation purposes. More specifically, there will be one convolutional layer with 16 8×6 sized kernels (eight time frames, six mel bands, 16 output channels), followed by a max-pooling layer of size 3×6. The local feature extraction unit is finalised by another convolutional layer with 32 6×3 sized kernels. On top, we apply a dropout layer with 128 connections (a fully connected layer with 128 units trained with the dropout method), followed by a binary unit, whose dropout probability was set to $p = 0.5$. Due to time constrains and the enormous search space spanned by the hyper-parameters, an investigation of all hyper-parameters is not feasible.

3.3. Training

For the network training, we utilise the former described spectrogram excerpts, the binary target labels and a set of corresponding weights. To face the problem of annotation inaccuracy as described in the introduction

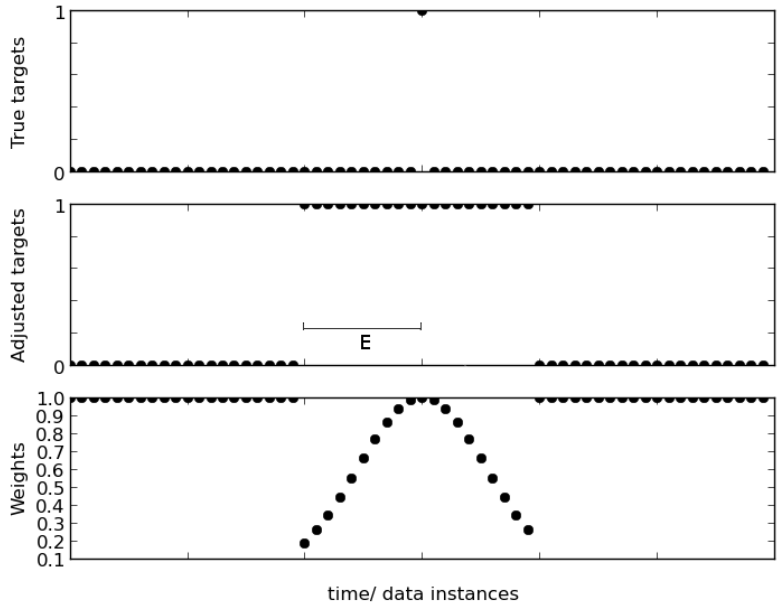

Figure 3.1: The top panel shows an annotated segment boundary. Each marker represents one time frame. The middle panel displays the adjusted target labels. The target environment is indicated by an E. The targets are set to one in the environment of this boundary, and to zero elsewhere. The bottom panel illustrates the weights corresponding to each time frame. Note that the weights in the target environment have weakened weights for the training process.

we introduce a concept we call *target smearing*. For that purpose we adapt Equation 2.16 as follows:

$$E(\theta) = -\sum_{n=1}^{N} q_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}, \tag{3.1}$$

in which q_n denotes the weights corresponding to each frame. Each weight is determined as follows: First we set all target labels in an environment of $\pm E$ (*target environment*) around the annotated segment boundary to one. All other labels will be zero. Consequently, we prune the weights in the target environment by a Gaussian function as illustrated in Figure 3.1 for $E = 10$. Weights that are not influenced by the Gaussian are set to 1. Initially, we also experimented with a triangle and a rectangular function with less good results. A second concern is that positive and negative examples are not balanced that is, there are much more negative than positive ones. To account for that we feed each positive target three times into the data set (*boundary pushing*).

We train the network with backpropagation based stochastic gradient descent on the error function Equation 3.1. We utilise mini-batches of size 64, a learning rate of 0.6,learning-rate decay of 0.6 (according to Equation 2.29), momentum of 0.95. Each iteration is completed after 2000 weight updates (one epoch). We always perform 20 epochs, because we watched the validation set error to stagnate after. We trained with theano [10] on a GTX 780 Ti.

3.4. Network variability

Training two networks with the same hyper-parameter configuration with different initial weights results in different networks, that may have different output when providing the same input. Thus also the network

performance assigned to a configuration will vary from case to case. In order to be able to still make conclusions about hyper-parameter settings. We assume the network performance to be a Normal distributed random variable. To that end, for every experiment, we train 5 networks with the same configuration. To compare configurations we will check differences of performances (F-measure) at a level of confidence of 0.95%. To further support generalisation we will average the model output, so-called *bagging*, and also provide the corresponding performance result.

3.5. Boundary prediction from network output

After training, networks are applied to pieces of music. For every time frame, the network computes an output, which can be interpreted as the probability to be a boundary point for the particular frame. By concatenating all frames, we obtain a curve for the entire musical piece, in which the ordinates that have the highest probability within a local environment, constitute predicted boundary points (*peak-picking*). More specifically, every output value that is not surpassed within ± 6 seconds is a boundary candidate. In order to compensate for long-term trends, we subtract the average of the activation curve in the past twelve and the upcoming six seconds from each candidate value. An example for a network output curve, a set of predicted and annotated boundaries can be found in Figure 2.12. Consequently, we obtain a list of potential boundaries, which is further specified by means of a threshold, whose optimum is computed on the validation set and adjusts the sensitivity of the classifier.

chapterExperimental results and analysis This chapter is split into three sections. Before we run large-scale experiments to compare and optimise various hyper-parameters, we will report on preliminary investigations on several aspects of the classifier creation process. Finally, we will infer properties of the trained networks by visualising which input regions have highest influence on the output. As explained in Section 2.3.1, we rely on the F-measure with tolerances $\tau = \pm 0.5s, \pm 3s$ for network performance evaluation.

3.6. Preliminary considerations

We performed several preliminary experiments to determine choices and ranges in the present experiments. Specifically, the current model skeleton (see Section 3.2) was designed after exploring a simplified artificial data set. We called this set *texture mix* because we faded various sound textures into each other to obtain simple boundaries. Moreover, we examined the last layers, peak-picking procedure and the bounds for the network performance more closely.

3.6.1. Network performance with dropout

Initially, we were concerned with the two layers that influenced the output of a network the most: the fully connected layers of the architecture. According to recent literature, the dropout method is to be preferred over not applying this technique [81]. In contrast to most applications, however, we utilise our NN only to compute a mid-level representation of the data. As a consequence, a probability curve is created that needs to be post-processed. As a consequence of this speciality, we investigated the performance of the network when applying fully connected with and with-out dropout to the networks's last layers L and $(L-1)$. Dropout layers (as

defined in Section 2.2.3.2) performed best for both layers so that we will fix our training method to use the dropout technique.

3.6.2. Peak-picking optimisation

As explained in Chapter 3, an input spectrogram excerpt delivers a probability estimate for one time frame to be a boundary. Concatenating the time frames leaves us with a curve that needs to be peak-picked for sakes of boundary prediction. Initially, we applied 4 different peak-picking methods on the resulting probability curves and consequently decided which one to use henceforth. The peak-pickers comprise various parameters that need to be tuned. The most important one is the threshold which a predicted boundary must surpass. Note that the optimal threshold is different for every trained network. Our aim in this study is to optimise the threshold for the F-score regardless of the corresponding precision and recall. However, it would be possible to change the focus on balanced rates as well. In order to find the maximum F-score, we compute all F-measures on a

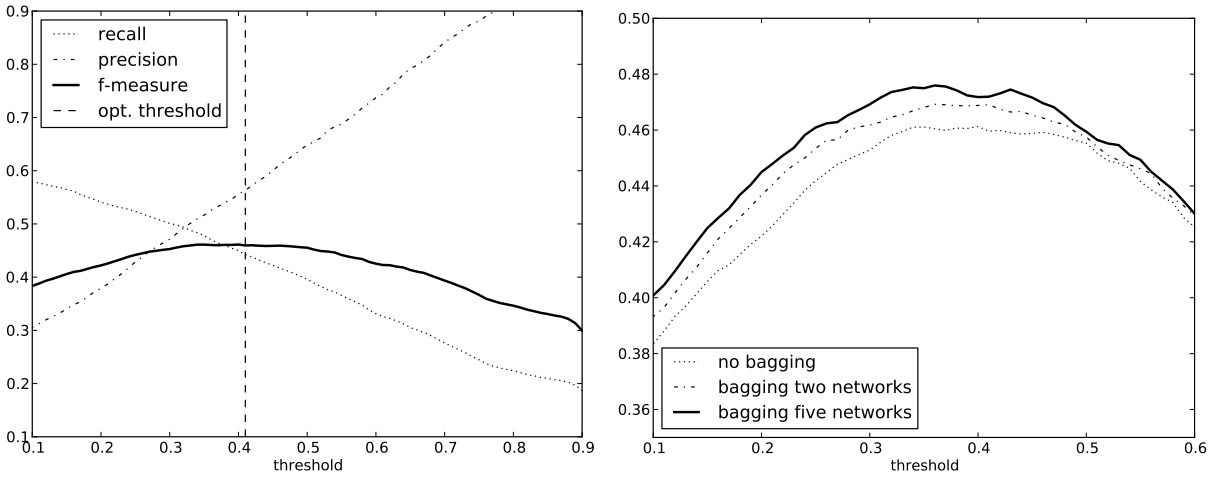

Figure 3.2: On the left we show precision, recall and F-measure for a randomly chosen model when evaluating it with tolerance of ± 0.5 over the validation set. The Optimal is indicated as a vertical line. On the right we show how the F-score curve changes when bagging multiple model of equal hyper-parameter configuration.

(non-equally spaced) grid of thresholds between 0.05 and 0.75 on our validation set (see Section 2.3.2). To justify the range and the number of thresholds, we computed an extremely fine-grained plot for one specific example of tolerance and network architecture, in which the dependence of F-measure precision and recall on the applied threshold is visible (see Figure 3.2). You can see a plateau around the optimal F-score (indicated by a vertical dashed line), suggesting the choice of the threshold not to be very sensitive. This gives us reason to believe that we can choose a proper threshold even with a weak validation set and coarse-grained threshold grids. When Bagging networks, however, the results could get more sensitive to the threshold choice. When inspecting the right side of the Figure 3.2 we see that this is not the case when bagging up to 5 networks of equal hyper-parameter configuration.

3.6.3. Baseline and upper bound

At this point, we will determine a baseline and an upper bound for the accuracy of our networks. Theoretically, the F-measure is bounded by $F \in [0, 1]$. The upper bound is what can be achieved with perfect recall and

precision rate. Practically, however, this is not accomplishable for our task as we will show in the following. Instead, we will compute practically relevant bounds $[F_{\text{inf}}, F_{\text{sub}}]$. In the introduction we discussed that music segmentation is a complex task also for human interpreters. Thus when two experts structure the same song there may be different interpretations, despite concrete annotation guidelines. Naturally, we developed the idea to compare two human experts to see how well they score compared to each other. We therefore expect our algorithm not to score higher then when comparing two human experts. We will refer to the upper bound as F_{sup}. To compute F_{sup} we used 498 pieces of our data set that have been annotated twice. Consequently, we labelled one of the annotations as ground-truth and one as prediction. Due to the symmetry of the F-measure, it does not matter which one was the prediction and which one the ground-truth. The derived upper bounds are $F_{\text{sup},0.5s} = 0.67$ for $\tau = \pm 0.5s$ and $F_{\text{sup},3s} = 0.76$ for $\tau = \pm 3s$. One may object that ML algorithms can beat human recognition, e.g., in handwritten digit recognition. In contrast to this work, the authors of those studies trained the networks on the intended digit targets, not by targets annotated by a second person. In order to infer an approximate benchmark F_{inf} for a reasonable baseline algorithm, one that is more accurate than a well-chosen random choice of segment boundaries, we compute equidistant boundaries over our dataset. For each piece in the database the first and the last time frame were marked as boundaries. This is crucial because most pieces have annotated boundaries near the beginning and end. For optimal spacing, we achieve F-scores from $F_{\text{inf},0.5s} = 0.13$ for $\tau = \pm 0.5s$ and $F_{\text{inf},3s} = 0.33$ for $\tau = \pm 3s$.

3.7. Main experiments

In this section, we will describe experiments that were conducted to tune the hyper-parameters of our network. We will evaluate their performance on our validation set. Consequently, we will compute the F-score of the best hyper-parameters on the test set. A comparison of our results to recorded performances in the MIREX campaign 2012 and 2013 can be found in the discussion (see Chapter 4). The networks that we train have a large amount of hyper-parameters that need to be tuned, such as the network architecture or gradient descent related parameters. With the set of experiments we describe in this section, we will try to optimise these hyper-parameters manually as they can not be learned. In order to conduct only a reasonable amount of experiments, the following assumptions are made: We split the set of hyper-parameters into disjoint subsets we assume to be independent, and then optimise the hyper-parameter subsets one after another. The independence assumption may not hold in every case nor can the justification be proven. However, it is intuitive and due to the large search space and restricted amount of time (computing power) we are forced to opt for this strategy.

3.7.1. Optimisation of architectural hyper-parameters

In this set of experiments we will optimise the convolutional layers. For that purpose, we will inspect the time and frequency pooling between both convolutional layers and the number of kernels. We have chosen these hyper-parameters because they influence the model capacity the most. To determine the other hyper-parameters (see Section 3.5), we will use a base architecture that has been proven to perform well. More specifically, the architecture was set to one convolutional layer with 16 kernels of size 8×6 (eight time frames and six mel-scaled frequency frames) each, followed by a max pooling layer of size 3×6 and another convolutional layer with 32 kernels of size 6×3. For the classification, we applied two dropout layers with 128 and 1 connections. Note that parts of this architecture will vary for the sake of these experiments. The input is defined by 16 seconds

long spectrogram excerpts with high resolution and a target smearing of $2E = 1.5s$. The learning parameters are fixed as described in Section 3.2.

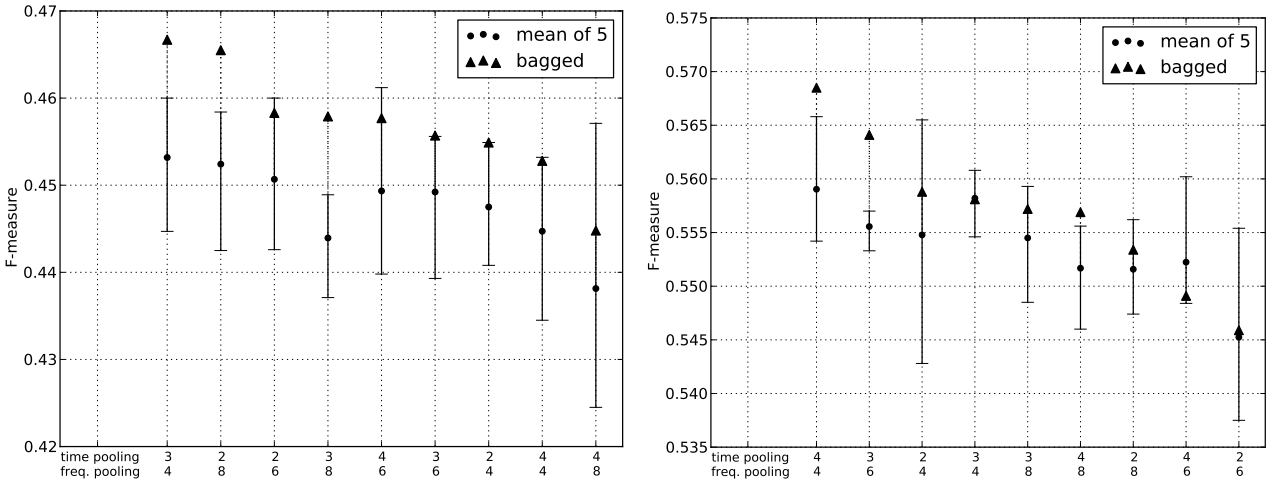

Figure 3.3: Network performance for various max pooling layer sizes. Specifically, we evaluated results for different layer sizes with respect to the pooled time frames (decoded as time pooling) and to the pooled mel-scaled frequency bands (decoded as freq. pooling). Left: F-measure with $\tau = \pm 0.5s$. Right: F-measure with $\tau = \pm 3s$. We trained 5 networks independently with random initialisation for each configuration. We indicated their mean, min-max-range and the result when bagging them.

Firstly, we vary the pooling layer sizes between the two convolutional layers. An important motivation for this experiment is that the pooling size crucially influences the runtime of the experiment. Large pooling grids would save considerable amounts of time. The time pooling will be tested for two, three and four time frames to be max pooled and the mel-scaled frequency pooling (freq. pooling) will be tested for the sizes four, six and eight. The results of this experiment are presented in Figure 3.3. The mean performance of five randomly initialized networks trained from different random initialisations is indicated, as well as the min-maximum range and the result of bagging all networks together. The hyper-parameter configurations are sorted by the best F-scores, and separated for both time tolerances. For exact information please examine Table 5.1 and 5.1 in the appendix. When inspecting the results, surprisingly, the size of the pooling grid range chosen has hardly any influence on the performance for the validation set. None of the conditions differs significantly from any other at a level of 0.95% reliance. There is only a trend that a frequency pooling of eight seems too large in both tolerance cases. A time pooling of two on the other hand, may be too small at least for $\tau = \pm 3s$. Similarly, for $\tau = \pm 0.5s$ four max pooled time frames may be too much. Excluding these values, we suggest a 3×6 sized pooling layer for optimal performance. We decided so despite the fact that 3×4 patches worked slightly better for $\tau = \pm 0.5$, because in that way we save computing time and we obtain consistency regarding both tolerances.

Secondly, we wanted to inspect whether a larger model contributes to a performance increase of our model. For that purpose, we varied the number of kernels in the first layer between eight and 16 and the ones in the second layer from 16 to 64, while keeping their size constant. In Figure 3.3 results for both time tolerances are visualised. Again we display mean, min-max range and the bagged model performance. The exact values can be found in Table 5.3 and Table 5.4 in the appendix. For both F-measures, we find eight kernel in the first layer to perform significantly worse than other options. We also find a tendency that 16 kernels work best. At

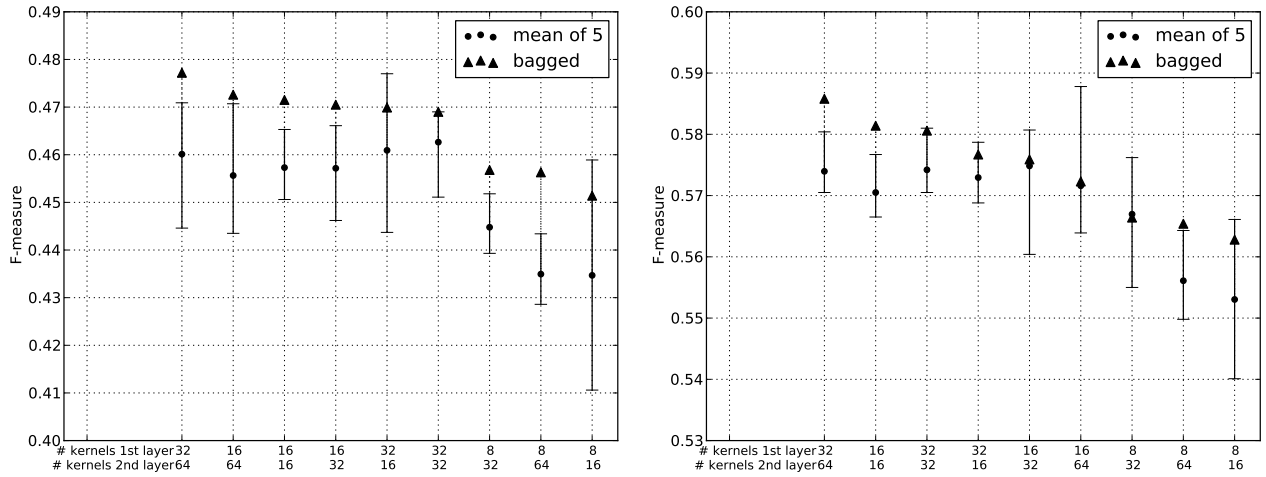

Figure 3.4: Network performance for various numbers of kernels for the first (decoded as # kernels 1st layer) and second (decoded as # kernels 2nd layer)convolutional layer. Left: F-measure with $\tau = \pm 0.5s$. Right: F-measure with $\tau = \pm 3s$. We trained 5 networks independently with random initialisation for each configuration. We indicated their mean, min-max-range and the result when bagging them.

$\tau = \pm 0.5s$ tolerance, there is a tendency for larger models, i.e., the second layer should have 64 kernels. In contrast to that, at $\tau = \pm 3s$ tolerance, smaller models tend to perform better. For both measures, 32 kernels in the first and 64 kernels in the second layer lead to optimal performance. Hence, we again obtain consistency.

All in all, these experiments show that the performance of the network is relatively robust regarding the architectural aspects that we investigated, which was not expected.

3.7.2. Optimisation of input related hyper-parameters

Apart from the architecture, there are other hyper-parameters: There is the set of hyper-parameters that defines the data input, e.g., the considered context length N or the applied smoother function, and furthermore, there is the set that defines the gradient descent search. We will only conduct experiments on the set of input defining parameters. For each of these experiments, we set the gradient search defining parameters as described in Section 3.3, assuming them to work well on all configurations. This hypothesis was extensively tested for many settings by checking the error-epoch curve. The focus on the considered input defining hyper-parameters arises from the expected relevance for the performance. In particular, we cross-optimised the temporal context, i.e., the length of the spectrogram excerpt, the temporal resolution and the temporal length of the applied target smearing. In the Appendix (Section 5.1) we listed all results. A selection of the most expressive ones is visualised in Figure 3.5 for $\tau = \pm 0.5s, \pm 3s$. Each ordinate relates to one hyper-parameter configuration, that is encoded in the table below. For the five computed networks per configuration we provide the mean, the maximum-minimum range and the result for bagging all five, sorted by performance. We see that some networks have a greater performance increase when bagging than others. In that way, clusters occurring differently for the two time tolerances are visible. At $\tau = \pm 0.5s$, a small target smearing is preferred, ideally it is $2E = 1.5s$. Context length and resolution seem to be equally important. These factors are both crucial for defining the model capacity.[1] Hence, it is not surprising that they are closely related. The best network can be found for

[1] Resolution and context length define the number of frames the networks sees as input and thus the size of the feature maps to be computed

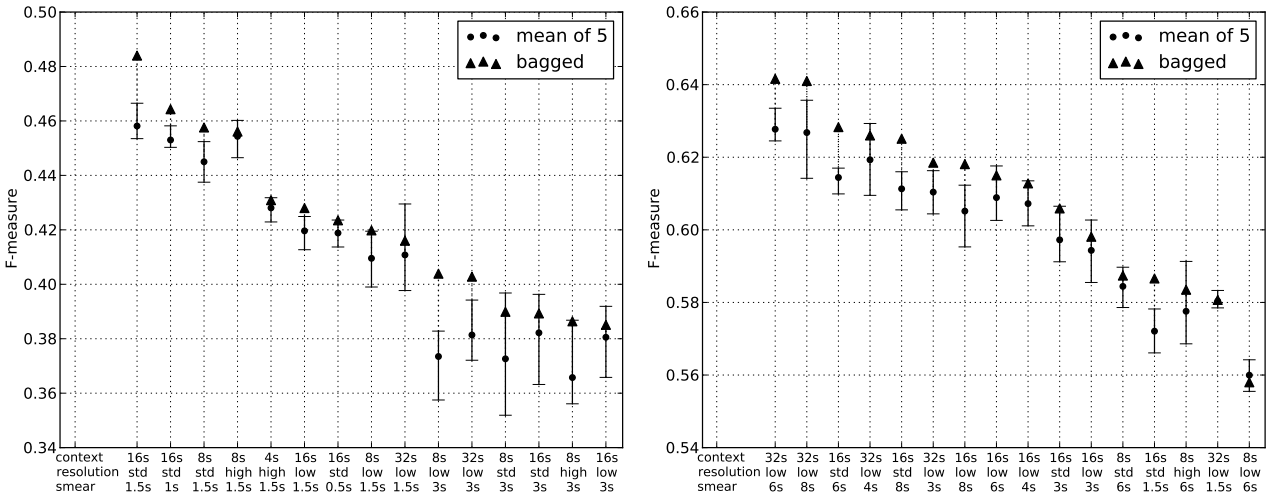

Figure 3.5: Network performance for various input defining hyper-parameter configurations. Specifically, we evaluated results for different resolutions (decoded as resolution), temporal lengths of spectrogram excerpts (decoded as context) and smearing environments $2E$ (decoded as smear). Left: F-measure with $\tau = \pm 0.5s$. Right: F-measure with $\tau = \pm 3s$.

a context of 16s, at standard resolution and a smearing of 1.5s with an F-score of 0.4840. For $\pm 3s$ accepted inaccuracy, we find that a large context adds to the performance (32s) as well as a large smearing (6s) while keeping the model capacity reasonably small with low resolution. We find the best network for a context of 32s, at low resolution and a smearing of 6s with an F-score of 0.6416. Finally we would like to point out that this experiment was one of the most time consuming ones with a total runtime of almost 14 days, with a single network training between 60 and 120 min.

3.7.3. Questioning the model

So far, we assumed that our networks benefit from exploring spatial relationships in the spectrograms that we referred to as feature learning. In this section, we will provide indications that this claim is indeed true. In order to do so, we will apply dropout layers on a highly developed feature that is designed to be invariant against octave transformations: the chroma vectors (see Section 2.1.1). In that context, they may also be called regularisers.

More specifically, we passed the data through one dropout layer with varying number of units and one binary output unit. In that manner the architecture is close to the previous experiments and still considers a potential change of the model's capacity. We also trained one network with two hidden dropout layers with each 2058 connections without remarkable difference in performance to other models. The results are shown in Figure 3.7.3 and 3.7.3. The best results are significantly below the ones that built features via convolution and approximately in the range of the current state-of-the-art methods (see Tables 4.1 and 4.2 for comparison). Thus, we can conclude: Our neural networks perform already well when being trained on very restricted input features such as chroma, outperforming methods that use hand-crafted high-level features. However, they perform worse than the models for which we apply a comparably simple feature (spectrum) and convolutional layers, indicating that it helps when the network can choose the feature representation itself. We also examined this experiment with MFCCs as well, with similar results.

$\tau = 0.5$	# connections in DL	bagged	mean	max	min
	8192	0.3242	0.3207	0.3262	0.3182
	2058, 2058	0.3227	0.3199	0.3241	0.315
	4096	0.3179	0.3145	0.3191	0.3108
	2058	0.3115	0.3122	0.3242	0.3034
	512	0.3092	0.3043	0.3123	0.3002
	64	0.3049	0.294	0.3009	0.2824
	256	0.2997	0.3013	0.3073	0.2949

Table 3.1: Results from convolution-free experiments (see Section 3.7.3). The table compares different model capacities. The table yields information about all experiments conducted, F-score for $\tau = \pm 0.5s$ on the validation set.The best result in this experiment is significantly below any result with convolutional layers.

$\tau = 3s$	# connections in DL	bagged	mean	max	min
	4096	0.5152	0.5052	0.5166	0.4982
	2058, 2058	0.5076	0.5036	0.5096	0.4992
	8192	0.5061	0.5076	0.5155	0.499
	2058	0.4992	0.5027	0.5052	0.4998
	512	0.4861	0.4891	0.4939	0.4865
	64	0.4829	0.4815	0.4887	0.4716
	256	0.4816	0.4815	0.4888	0.4701

Table 3.2: Results from convolution-free experiments (see Section 3.7.3). The table compares different model capacities. The table yields information about all experiments conducted, F-score for $\tau = \pm 3s$ on the validation set.The best result in this experiment is significantly below any result with convolutional layers.

3.8. Feature importance and network investigation by visualisation

Following Simonyan et al. [76], we computed saliency maps for each particular spectrum excerpt. Following that, two different methods were applied to concatenate the excerpts.

Firstly, we added up all spectrogram excerpts' saliency maps that correlate to predicted boundaries. We did that for all songs of one network. We refer to it as *input saliency maps*. We visualised six different networks in Figure 3.6. We hope to see whether the particular networks used the entire context they were provided with and also to identify regions of high interest for boundary detection. If, for instance, there was more emphasis on post boundary events, we could change our model, to the effect that it considers more future than past per spectrum excerpt. In the first row we show networks trained with eight seconds (referred to as network 1) and 16 seconds (refereed to as network 2) long spectrum excerpts. When comparing these patterns, we find many similarities. We also note that the pattern of network 1 seems to be the stretched version of network 2. In the second row, we displayed two networks with equal hyper-parameters but different initialisation (referred to as networks 3 and 4). They are also very similar to each other. Note that the difference between network 2, and network 3 and 4 is the smearing. The less smearing in network 2 leads to a sharper pattern. Finally, we also display networks with 32 seconds considered input context (last row). Both perform best when applying the

F-measure with $\pm 3s$ tolerance. We see the more intense shading of these networks in comparison to all other networks. This is not surprising, because these networks perform best for $\tau = \pm 3s$ and thus allow naturally more boundaries than the more sensitive measure. Hence, we simply add more spectrum excerpts causing the more intense shading. In all networks, we find a similar pattern with two regions that have strong influence on the class decision, both of which are centred. In particular, we speak about the region of mel bands 45-55 which correlates to the frequencies 3.5-5.5 kHz and low frequency bands 0-5 according to 80-250 Hz. In the higher bands we can find the "t" and "s" sound of the human voice, a wide range of melody instruments (piano, violin, guitar, flute) but also percussion instruments like snares. In the lower region we find the ground tones of human voice and some bass instruments. Note also that the bands are relatively far apart from each other. That could indicate that they hold mostly separate information (basis vectors) for this problem.

Secondly, we added all frames corresponding to their actual temporal position (adding the overlapping between saliency maps), we call this the saliency song map. A number of randomly chosen song maps and their corresponding network output can be inspected in Figure 3.7. The purpose of this mapping is to identify different kinds of boundaries, e.g., could the network detect a boundary by different frequency bands. In fact, we can identify both high influence spots being more or less present when detecting boundaries. However certain "types" of boundaries can not be distinguished.

3.9. Summary

In order to build a final model for comparison to other methods, we summarise optimal model configurations based on our experiments. We will evaluate two variants optimised for $\tau = \pm 0.5s$ and $\tau = \pm 3s$ on our test set to which we refer to as $\mathbf{CNN}_{0.5s}$ and $\mathbf{CNN}_{3s}$. The former is provided with 8s of input on each side of the boundary (context=16s) while having standard resolution, whereas we provide the latter double the context at low resolution. Consequently, both networks have 116×80 sized inputs (116 time frames and 80 mel bands). The details of the feature extraction procedure are described in Section 3.1. The networks first convolutional layer contains 32 8×6 kernels and the second layer contains 64 3×6 kernels. The pooling layer in their middle will be of size 3×6. For training, we set the smearing 2E to 1.5s for $\tau = \pm 0.5s$ and 6s for $\tau = \pm 5s$. The training is performed as explained in Section 3.3. The final classifier is considered to be the bagged result of five trained networks.

Furthermore, we provide the result on the test set for our fully connected MLP trained on chroma vectors (see Section 3.7.3). The expected best model for $\pm 3s$ contains 4096 units and for $\pm 0.5s$ 8192. With the same logic as before we will refer to them as $\mathbf{Chroma}_{0.5s}$ and $\mathbf{Chroma}_{3s}$.

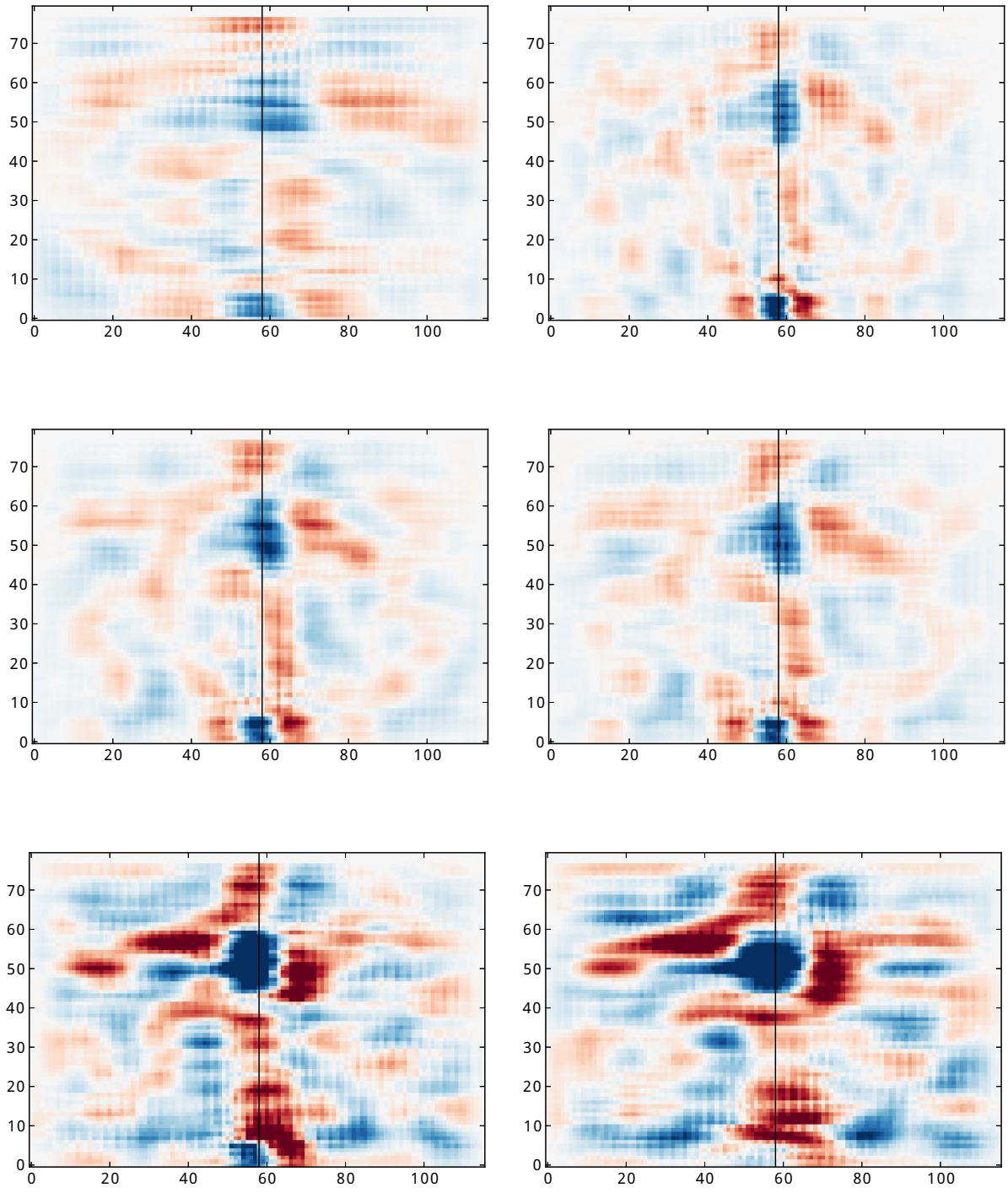

Figure 3.6: We visualised the sum of all time frames of all songs for the most successful networks. Top: Left, a network with 8s of context (1). Right, a network with 16s context (2). Middle: Two networks with 16s context (3,4), same parametrisation but different initialisation. The differences are minimal. Note also that network (2) has a smaller smearing. That seems to influence the sharpness of the centre region. Bottom: Best networks for a tolerance of $\pm 3s$. Both have 32s context length (5,6). Note that the shading here is generally darker because those models can and optimally will predict more boundaries and thus we simply added more maps.

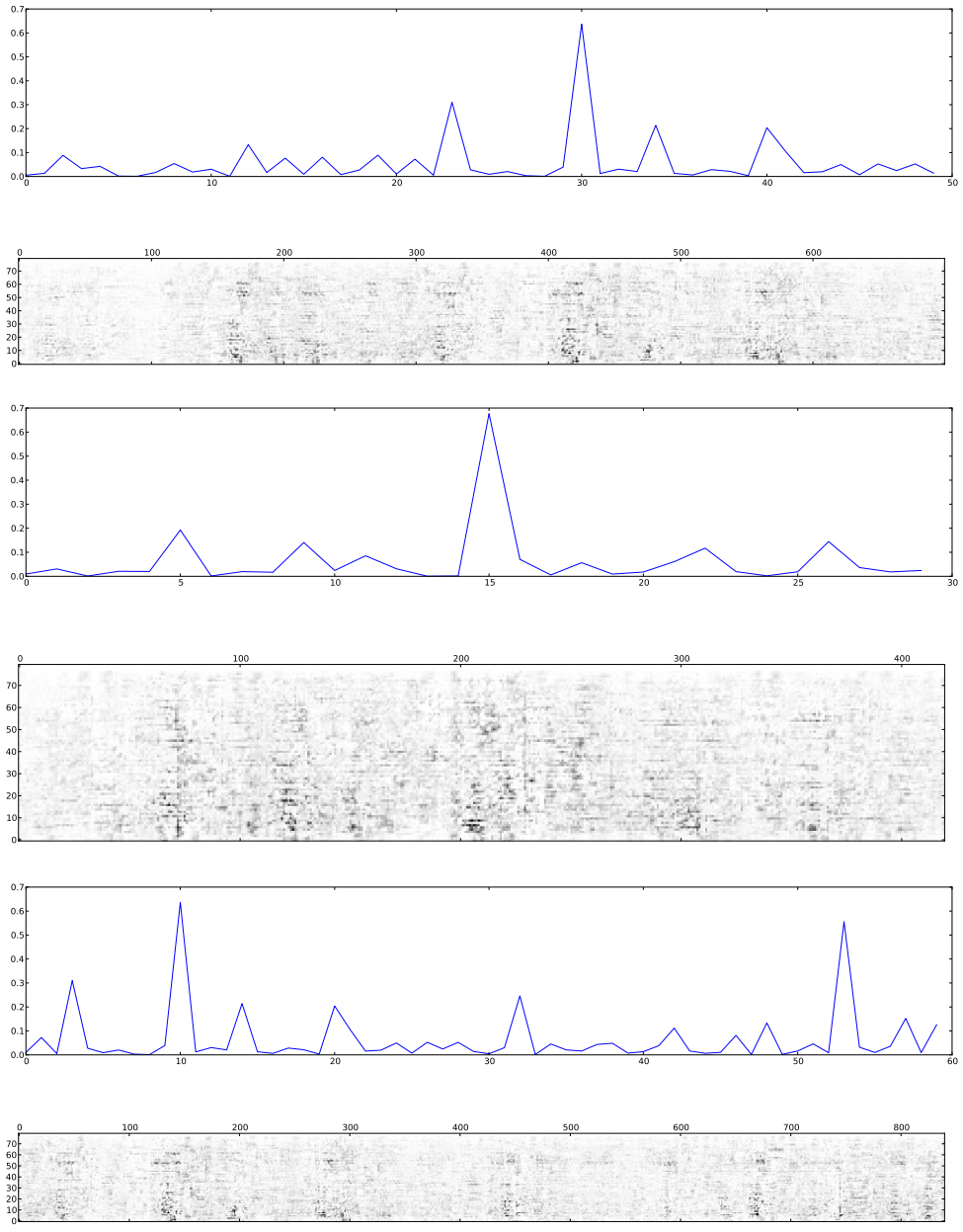

Figure 3.7: Three regions of high output activity from three randomly chosen songs from the SALAMI data set. On top we show the network output, on the bottom of each pair the song saliency map. We can try to relate peaks to the saliency input maps in Figure 3.6.

4. Discussion and outlook

In this study we developed a comprehensive system to find segment boundary points in musical data, that is strongly outperforming existing methods. With respect to the objectives we defined in the introduction, we found our classifier to train well on MFCCs, chroma and mel spectrograms. While MFCCs and chroma vectors are highly developed features of low dimensionality, we found our classifiers to learn best on the simple high dimensional mel spectrograms. This result is comparable to findings in computer vision, where simple features also are preferred over more complex ones. Furthermore, we engineered an FNN classifier that fits the needs of our data. In particular, we trained on human-annotated data in a supervised fashion. For that purpose, we adapted and further developed the method from Schlüter and Böck [72], who apply it for onset detection. Considering data-driven models, the chosen classifier holds many advantages. In particular, it can handle large amounts of data especially well by performing online learning and can deal with temporal target uncertainty by an adjusted binary-cross entropy objective (see Equation 3.1). Furthermore, it allows us to explore spatial relationships by weight constraints in the convolutional layers. In that context, we also experimented with different model regularises. Here, we showed two model regularisers, namely, convolutional layers and hand-crafted chroma features to perform well on the task. The best results were achieved with a network with two convolutional layers and two fully connected ones. In that context, we calculated an approximate lower and upper bound we expect all algorithms to fall in. The achieved F-scores outperform all algorithms submitted to MIREX 2012 and 2013 (see Table 4.1 and 4.2). Moreover, through visualisation we identified frequency bands of high interest for boundary detection (see Section 4.5). We also showed that although a network uses mainly the information close to the boundary it does use the provided context as well. Thus, contextual information does play a role in boundary decision making.

Next, we will name approaches that could improve the currently applied method further. First, we would like to be able to apply more complex architectures in order to feed the network multiple features at the same time. That means, for example, that we may train convolutional layers on spectrogram excerpts and one fully connected one on corresponding onsets. Subsequently, we feed both feature maps to one fully connected layer. For that purpose, we need to extend our current framework. In that context we would also like to implement recent developed deep learning techniques such as maxout [31]. Secondly, with respect to the structural hierarchy of most pieces of music mentioned in the introduction, we can adjust our training method in such a way that the network trains higher level boundaries more intensely than lower level ones, i.e., by adjusting the weights or the number of times a boundary is fed into the data set (see Section 3.3). We expect that this would not only improve current results but also allow us to predict boundaries at different hierarchical levels. Additionally, bagging different parametrisations may boost the method's performance further. Thirdly, we only explored hyper-parameter settings by grid search. A more advanced search algorithm could help to make further improvements. In fact, we already developed a CMA-ES based algorithm to tune hyper-parameters.

Unfortunately, due to time constraints we could not provide experiments yet. Finally, the classifier performance trained on spectrograms of raw audio recordings may suffer from loudness differences, and background noise in the beginning and end of recordings. Especially, the latter one may be a problem to the network because the abrupt change of our pink-noise to the song's specific background-noise may lead to false detections. We can fix loudness differences by adjusting the loudness in the entire audio file or by using a dynamic gain control. For the padding, we need to think of a more intelligent way to provide background in the beginning and the end of recordings. It must be able to relate to the "natural" background of every individual recording. These pre-processing steps may currently occupy some of our model capacity that we may reclaim in that manner. Finally, note that our framework can also be applied to other tasks in MSA such as for a different foci of annotation or for song boundary detection.

$\tau = 0.5$	Algorithm	F-measure	Precision	Recall
	Upper bound (est.)	0.68		
	CNN$_{0.5s}$	**0.4756**	0.5724	0.4633
	Chroma$_{0.5s}$	**0.3349**	0.4302	0.3237
	MP2 (2013)	0.3280	0.3001	0.4108
	MP1 (2013)	0.3149	0.3043	0.3605
	OYZS1 (2012)	0.2899	0.4561	0.2583
	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
	CNN$_{3s}$	**0.2663**	0.3316	0.2452
	RBH1 (2013)	0.2567	0.2043	0.3936
	RBH2 (2013)	0.2567	0.2043	0.3936
	RBH4 (2013)	0.2567	0.2043	0.3936
	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 4.1: Final results of this work in comparison with current state-of-the-art algorithms (MIREX campaign in 2012 and 2013). We emphasised our best result derived by various experiments that were evaluated on the validation set. Specifically, **CNN**$_{0.5s}$ is the expected best convolutional model for $\tau = \pm 0.5$, **CNN**$_{3s}$ for $\pm 3s$ respectively, and **Chroma**$_{0.5s}$ the expected best convolution-free model using chroma for $\tau = \pm 0.5$. Results were computed on the test set.

$\tau = 3s$	Algorithm	F-measure	Precision	Recall
	Upper bound (est.)	0.76		
	CNN$_{3s}$	**0.6185**	0.5866	0.7214
	CNN$_{0.5s}$	**0.5795**	0.5714	0.6734
	MP2 (2013)	0.5213	0.4793	0.6443
	MP1 (2013)	0.5188	0.5040	0.5849
	CF5 (2013)	0.5052	0.3990	0.7862
	Chroma$_{3s}$	**0.5015**	0.4826	0.6242
	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
	OYZS1 (2012)	0.4401	0.6354	0.4038
	SBV1 (2012)	0.4352	0.3694	0.5929
	MHRAF1 (2012)	0.4192	0.4342	0.4447
	Baseline (est.)	0.33		

Table 4.2: Final results of this work in comparison with current state-of-the-art algorithms (MIREX campaign in 2012 and 2013). We emphasised our best result derived by various experiments that were evaluated on the validation set. Specifically, **CNN**$_{0.5s}$ is the expected best convolutional model for $\tau = \pm 0.5$, **CNN**$_{3s}$ for $\pm 3s$ respectively, and **Chroma**$_{3s}$ the expected best convolution-free model using chroma for $\tau = \pm 3$. Results were computed on the test set.

5. Appendix

5.1. Experimental Results

$\tau = 05s$	time pooling	freq. pooling	bagged	mean of 5	max	min
	3	4	0.4667	0.4532	0.4600	0.4447
	2	8	0.4655	0.4524	0.4584	0.4425
	2	6	0.4583	0.4507	0.4600	0.4426
	3	8	0.4579	0.4439	0.4489	0.4371
	4	6	0.4577	0.4493	0.4612	0.4398
	3	6	0.4557	0.4492	0.4556	0.4393
	2	4	0.4549	0.4475	0.4549	0.4408
	4	4	0.4528	0.4447	0.4532	0.4345
	4	8	0.4448	0.4381	0.4571	0.4245

Table 5.1: Results from the pooling layer size determining experiments in Section 3.7.1. The table yields information about best the best results for $\pm 0.5s$ tolerance on the validation set.

$\tau = 3s$	time pooling	freq. pooling	bagged	mean of 5	max	min
	4	4	0.5685	0.5590	0.5658	0.5542
	3	6	0.5641	0.5556	0.5570	0.5533
	2	4	0.5588	0.5548	0.5655	0.5428
	3	4	0.5581	0.5582	0.5608	0.5546
	3	8	0.5572	0.5545	0.5593	0.5485
	4	8	0.5569	0.5517	0.5556	0.5460
	2	8	0.5534	0.5516	0.5562	0.5474
	4	6	0.5491	0.5522	0.5602	0.5484
	2	6	0.5459	0.5452	0.5554	0.5375

Table 5.2: Results from the pooling layer size determining experiments in Section 3.7.1. The table yields information about best the best results for $\pm 3s$ tolerance on the validation set.

$\tau = \pm 0.5s$	# kernels 1st layer	# kernels 2nd layer	bagged	mean of 5	max	min
	32	64	0.4772	0.4601	0.4709	0.4446
	16	64	0.4726	0.4556	0.4707	0.4435
	16	16	0.4715	0.4573	0.4653	0.4506
	16	32	0.4705	0.4572	0.4661	0.4462
	32	16	0.4699	0.4609	0.4770	0.4437
	32	32	0.4690	0.4626	0.4690	0.4511
	8	32	0.4568	0.4448	0.4518	0.4393
	8	64	0.4563	0.4349	0.4434	0.4286
	8	16	0.4514	0.4347	0.4589	0.4106

Table 5.3: Results from architectural experiments regarding the number of kernels (see Section 3.7.1). The table yields exact information about all experiments conducted, F-score for $\tau = \pm 0.5s$ on the validation set.

$\tau = \pm 3s$	# kernels 1st layer	# kernels 2nd layer	bagged	mean of 5	max	min
	32	64	0.5858	0.5740	0.5804	0.5705
	16	16	0.5814	0.5705	0.5767	0.5665
	32	32	0.5806	0.5742	0.5810	0.5705
	32	16	0.5767	0.5729	0.5787	0.5688
	16	32	0.5759	0.5748	0.5807	0.5604
	16	64	0.5723	0.5716	0.5878	0.5639
	8	32	0.5664	0.5670	0.5762	0.5550
	8	64	0.5654	0.5561	0.5643	0.5498
	8	16	0.5628	0.5530	0.5661	0.5401

Table 5.4: Results from architectural experiments regarding the number of kernels (see Section 3.7.1). The table yields exact information about all experiments conducted, F-score for $\tau = \pm 3s$ on the validation set.

$\tau = 0.5s$	context length N	resolution	smearing 2E	bagged	mean	max	min
	16s	std	1.5s	0.4840	0.4581	0.4665	0.4535
	16s	std	1s	0.4643	0.4530	0.4582	0.4503
	8s	std	1.5s	0.4576	0.4450	0.4524	0.4375
	8s	high	1.5s	0.4561	0.4543	0.4602	0.4465
	4s	high	1.5s	0.4309	0.4281	0.4318	0.4229
	16s	low	1.5s	0.4280	0.4196	0.4249	0.4127
	16s	std	0.5s	0.4235	0.4188	0.4236	0.4137
	8s	low	1.5s	0.4198	0.4095	0.4195	0.3990
	32s	low	1.5s	0.4160	0.4108	0.4295	0.3977
	8s	low	3s	0.4039	0.3735	0.3828	0.3575
	32s	low	3s	0.4028	0.3814	0.3942	0.3721
	8s	std	3s	0.3899	0.3726	0.3968	0.3519
	16s	std	3s	0.3893	0.3822	0.3963	0.3632
	8s	high	3s	0.3864	0.3657	0.3868	0.3561
	16s	low	3s	0.3851	0.3805	0.3919	0.3658
	4s	high	3s	0.3646	0.3485	0.3759	0.3170
	32s	low	4s	0.3589	0.3231	0.3360	0.3041
	16s	low	4s	0.3514	0.3318	0.3537	0.3146
	32s	low	6s	0.2967	0.2675	0.2934	0.2539
	16s	low	6s	0.2547	0.2455	0.2723	0.2216
	16s	std	6s	0.2416	0.2380	0.2515	0.2302
	32s	low	9s	0.2396	0.2109	0.2281	0.1947
	8s	std	6s	0.2349	0.2202	0.2327	0.1954
	8s	low	6s	0.2246	0.2201	0.2418	0.2077
	8s	high	6s	0.2136	0.2199	0.2454	0.2019
	16s	std	9s	0.2068	0.1945	0.2198	0.1644
	16s	low	9s	0.2006	0.1908	0.2001	0.1707

Table 5.5: Results from input optimizing experiments (see Section 3.7.2). The table yields information about all experiments conducted, F-score for $\tau = \pm 0.5s$ on the validation set.

$\tau = 3s$	context length N	resolution	smearing 2E	bagged	mean	max	min
	32s	low	6s	0.6416	0.6277	0.6335	0.6245
	32s	low	9s	0.6410	0.6268	0.6357	0.6142
	16s	std	6s	0.6283	0.6144	0.6170	0.6099
	32s	low	4s	0.6260	0.6193	0.6293	0.6095
	16s	std	9s	0.6251	0.6113	0.6160	0.6055
	32s	low	3s	0.6185	0.6104	0.6163	0.6044
	16s	low	9s	0.6181	0.6052	0.6123	0.5953
	16s	low	6s	0.6150	0.6089	0.6176	0.6026
	16s	low	4s	0.6128	0.6072	0.6135	0.6011
	16s	std	3s	0.6059	0.5972	0.6065	0.5912
	16s	low	3s	0.5981	0.5943	0.6027	0.5855
	8s	std	6s	0.5874	0.5844	0.5897	0.5786
	16s	std	1.5s	0.5866	0.5721	0.5782	0.5661
	8s	high	3s	0.5852	0.5739	0.5821	0.5656
	8s	high	6s	0.5835	0.5776	0.5913	0.5686
	32s	low	1.5s	0.5807	0.5804	0.5833	0.5785
	16s	low	1.5s	0.5750	0.5704	0.5755	0.5651
	8s	std	3s	0.5720	0.5640	0.5766	0.5505
	16s	std	1s	0.5668	0.5580	0.5668	0.5482
	8s	low	3s	0.5618	0.5546	0.5676	0.5480
	8s	low	6s	0.5580	0.5600	0.5642	0.5555
	8s	high	1.5s	0.5561	0.5529	0.5593	0.5448
	8s	std	1.5s	0.5549	0.5500	0.5598	0.5446
	8s	low	1.5s	0.5418	0.5365	0.5463	0.5292
	4s	high	3s	0.5376	0.5342	0.5538	0.5240
	4s	high	1.5s	0.5286	0.5183	0.5234	0.5095
	16s	std	0.5s	0.5275	0.5267	0.5331	0.5221

Table 5.6: Results from input optimizing experiments (see Section 3.7.2). The table yields information about all experiments conducted, F-score for $\tau = \pm 3s$ on the validation set.

Bibliography

[1] Large scale visual recognition challenge 2012 (ilsvrc2012). URL http://www.image-net.org/challenges/LSVRC/2012/results.html.

[2] Structural segmentation challenge. URL http://www.music-ir.org/mirex/2009/index.php/Structural_Segmentation.

[3] Music information retrieval evaluation exchange, 04 2014. URL http://www.music-ir.org/mirex.

[4] Salami data, 05 2014. URL http://ddmal.music.mcgill.ca/datasets/salami/SALAMI_data_v1.2.zip.

[5] S. Abdallah, M. Sandler, C. Rhodes, and M. Casey. Using duration models to reduce fragmentation in audio segmentation. *Mach. Learn.*, 65:485–515, 2006.

[6] J.-J. Aucouturier and M. Sandler. Segmentation of musical signals using hidden markov models. In *Proc. AES 110th Convention*, May 2001.

[7] L. Barrington, A. B. Chan, and G. Lanckriet. Modeling music as a dynamic texture. *Trans. Audio, Speech and Lang. Proc.*, 18:602–612, 2010.

[8] M. A. Bartsch and G. H. Wakefield. Audio thumbnailing of popular music using chroma-based representations. *Trans. Multi.*, 7(1):96–104, Feb. 2005.

[9] J. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler. A tutorial on onset detection in music signals. *Speech and Audio Processing, IEEE Transactions on*, 13:1035–1047, 2005.

[10] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proc. of the Python for Scientific Computing Conference (SciPy)*, June 2010.

[11] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[12] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Audio chord recognition with recurrent neural networks. In *Proc. ISMIR 14*, Nov 2013.

[13] M. J. Bruderer, M. McKinney, and A. Kohlrausch. Structural boundary perception in popular music. In *Proc. of 7th International Conference on Music Information Retrieval*, 2006.

[14] A. T. Cemgil, B. Kappen, P. Desain, and H. Honing. On tempo tracking: Tempogram representation and kalman filtering. *Journal of New Music Research*, 28:259–273, 2001.

[15] W. Chai. Automated analysis of musical structure. Master's thesis, Massachusetts Institute of Technology. Dept. of Architecture. Program In Media Arts and Sciences, 2005.

[16] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski. Computers beat humans at single character recognition in reading based human interaction proofs (hips. In *In 2nd Conference on Email and Anti-Spam*, 2005.

[17] M. Cooper and J. Foote. Summarizing popular music via structural similarity analysis. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 127–130, Oct 2003.

[18] G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton. Phone recognition with the mean-covariance restricted Boltzmann machine. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 469–477. 2010.

[19] S. Dixon, E. Pampalk, and G. Widmer. Classification of dance music by periodicity patterns, 2003.

[20] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003. ISBN 3540401849.

[21] A. Eronen. Chorus detection with combined use of mfcc and chroma features and image processing filters. In *In Proc. of the 10th Int. Conference on Digital Audio Effects (DAFx-07*, 2007.

[22] R. Fletcher. *Practical Methods of Optimization; (2Nd Ed.)*. Wiley-Interscience, 1987.

[23] J. Foote. Visualizing music and audio using self-similarity. In *Proceedings of the Seventh ACM International Conference on Multimedia (Part 1)*, MULTIMEDIA '99, pages 77–80, 1999.

[24] J. Foote. Automatic audio segmentation using a measure of audio novelty. In *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, volume 1, pages 452–455 vol.1, 2000.

[25] J. Foote and S. Uchihashi. The beat spectrum: a new approach to rhythm analysis. In *Multimedia and Expo, 2001. ICME 2001. IEEE International Conference on*, pages 881–884, Aug 2001.

[26] S. Gao, N. Maddage, and C.-H. Lee. A hidden markov model based approach to music segmentation and identification. In *Information, Communications and Signal Processing, 2003 and Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on*, volume 3, pages 1576–1580 vol.3, Dec 2003.

[27] P. E. Gill, W. Murray, and M. H. Wright. *Practical optimization*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1981. ISBN 0-12-283950-1.

[28] J. Goldberger, S. Gordon, and H. Greenspan. An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 487–493 vol.1, Oct 2003.

[29] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

[30] E. Gómez. *Tonal Description of Music Audio Signals*. PhD thesis, Universitat Pompeu Fabra, 2006.

[31] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. In *ICML (3)*, volume 28 of *JMLR Proceedings*, pages 1319–1327. JMLR.org, 2013.

[32] M. Goodwin and J. Laroche. A dynamic programming approach to audio segmentation and speech/music discrimination. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, volume 4, pages iv–309–iv–312 vol.4, May 2004.

[33] M. Goto. A chorus section detection method for musical audio signals and its application to a music listening station. *Audio, Speech, and Language Processing, IEEE Transactions on*, 14(5):1783–1794, Sept 2006.

[34] P. Grosche and M. Müller. A mid-level representation for capturing dominant tempo and pulse information in music recordings. In K. Hirata, G. Tzanetakis, and K. Yoshii, editors, *ISMIR*, pages 189–194. International Society for Music Information Retrieval, 2009.

[35] P. Grosche, M. Muller, and F. Kurth. Cyclic tempogram;a mid-level tempo representation for musicsignals. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 5522–5525, March 2010.

[36] N. Hansen. *The CMA Evolution Strategy: A Comparing Review*. Springer Berlin Heidelberg, 2006.

[37] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, (1):193–218.

[38] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke. Application of pretrained deep neural networks to large vocabulary speech recognition. In *Proceedings of Interspeech 2012*, 2012.

[39] K. Jensen. Multiple scale music segmentation using rhythm, timbre, and harmony. *EURASIP Journal on Advances in Signal Processing*, 2007(1):073205, 2007.

[40] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):221–231, Jan 2013.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1106–1114. 2012. URL `http://books.nips.cc/papers/files/nips25/NIPS2012_0534.pdf`.

[42] Y. LeCun, B. Boser, J. S. Denker, R. E. Henderson, D. band Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989.

[43] 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 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[44] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and M. K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[45] M. Levy and M. Sandler. Structural segmentation of musical audio by constrained clustering. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):318–326, Feb 2008.

[46] M. Levy, M. Sandler, and M. Casey. Extraction of high-level musical structure from audio data and its application to thumbnail generation. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 5, pages V–V, May 2006.

[47] T. L. Li, A. B. Chan, and A. H. Chun. Automatic musical pattern feature extraction using convolutional neural network. In *Proc. of the Int. MultiConf. of Engineers and Computer Scientists (IMECS)*, Hong Kong, Mar. 2010.

[48] LISA-lab. Deeplearning tutorials, Jul 2014. URL `http://www.deeplearning.net/tutorial/`.

[49] B. Logan. Mel frequency cepstral coefficients for music modeling. In *In International Symposium on Music Information Retrieval*, 2000.

[50] P. F. Lopes Carla. *Phone Recognition on the TIMIT Database - Chapter in Speech Technologies*. Ivo Ipsic, 2011. URL `http://www.intechopen.com/books/export/citation/BibTex/speech-technologies/phoneme-recognition-on-the-timit-database`.

[51] M. Marolt. A mid-level melody-based representation for calculating audio similarity. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 280–5, Victoria, Canada, 2006.

[52] B. McFee and D. Ellis. Learning to segment songs with ordinal linear discriminant analysis. In *International conference on acoustics, speech and signal processing*, ICASSP, 2014.

[53] M. Müller. *Information Retrieval for Music and Motion*. Springer, 2007.

[54] M. Müller and M. Clausen. Transposition-invariant self-similarity matrices. In *Proceedings of the 8th International Conference on Music Information Retrieval*, pages 47–50, Vienna, Austria, September 23-27 2007.

[55] M. Müller and F. Kurth. Enhancing similarity matrices for music audio analysis. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9–12, Toulouse, France, 2006.

[56] M. Müller and F. Kurth. Towards structural analysis of audio recordings in the presence of musical variations. In *EURASIP Journal on Advances in Signal Processing*, 2007.

[57] M. Müller, S. Ewert, and S. Kreuzer. Making chroma features more robust to timbre changes. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP '09, pages 1877–1880, Washington, DC, USA, 2009. IEEE Computer Society.

[58] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, Cambridge, MA, 2012.

[59] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[60] B. Ong. *Structural Analysis and Segmentation of Music Signals*. PhD thesis, University Pompeu Fabra, Barcelona, Spain, February 2007.

[61] 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 '06, pages 59–68, New York, NY, USA, 2006. ACM.

[62] J. Paulus and A. Klapuri. Acoustic features for music piece structure analysis. In *Conference: 11th International Conference on Digital Audio Effects (Espoo, Finland)*, 2008.

[63] J. Paulus and A. Klapuri. Music structure analysis using a probabilistic fitness measure and a greedy search algorithm. *Trans. Audio, Speech and Lang. Proc.*, 17:12, 2009.

[64] J. Paulus, M. Mueller, and A. Klapuri. Audio-based music structure analysis. In *International Society for Music Information Retrieval*, 2010.

[65] M. Peemen, B. Mesman, and H. Corporaal. Speed sign detection and recognition by convolutional neural networks.

[66] G. Peeters. Deriving musical structures from signal analysis for music audio summary generation: "sequence" and "state" approach. In U. Wiil, editor, *Computer Music Modeling and Retrieval*, volume 2771 of *Lecture Notes in Computer Science*, pages 143–166. Springer Berlin Heidelberg, 2004.

[67] G. Peeters. Template-based estimation of time-varying tempo. *EURASIP J. Appl. Signal Process.*, 2007 (1):158–158, Jan. 2007.

[68] G. Peeters. Sequence representation of music structure using higher-order similarity matrix and maximum-likelihood approach. In *Proceedings of the 8th International Conference on Music Information Retrieval*, pages 35–40, Vienna, Austria, 2007.

[69] C. Rhodes and M. Casey. Algorithms for determining and labelling approximate hierarchical self-similarity. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 41–6, Vienna, Austria, 2007.

[70] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.). Parallel distributed commnication. *The Bell System Technical Journal*, 27, 1986.

[71] SALAMI-Blog. About salami, 06 2014. URL `http://ddmal.music.mcgill.ca/research/salami/background`.

[72] J. Schlüter and S. Böck. Improved musical onset detection with convolutional neural networks. In *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[73] Y. Shiu, H. Jeong, and C.-C. J. Kuo. Musical structure analysis using similarity matrix and dynamic programming. In A. Vetro, C. W. Chen, C.-C. J. Kuo, T. Zhang, Q. Tian, and J. R. Smith, editors, *Multimedia Systems and Applications VIII*, volume 6015 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 398–409, 2005.

[74] Y. Shiu, H. Jeong, and C. J. Kuo. Similar segment detection for music structure analysis via viterbi algorithm. *2012 IEEE International Conference on Multimedia and Expo*, 0:789–792, 2006.

[75] P. Y. Simard, D. Steinkraus, and J. Platt. Best practice for convolutional neural networks applied to visual document analysis. In *In Proceedings International Conference on Document Analysis and Recognition (ICDAR)*, 2003.

[76] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[77] J. Smith. music structure and performance research. Music Computation and Cognition (MuCoaCo) Laboratory at the University of Southern California, 06 2014. URL `http://mucoaco.blogspot.co.at/2011/03/314-article-accepted-to-mcm.html`.

[78] 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 International Society for Music Information Retrieval Conference*, pages 555–560, 2011.

[79] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

[80] N. Srivastava. Improving neural networks with dropout. Master's thesis, University of Toronto, 2013.

[81] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and 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/v15/srivastava14a.html`.

[82] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[83] D. Turnbull and G. Lanckriet. A supervised approach for detecting boundaries in music using difference features and boosting. In *In Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, pages 42–49, 2007.

[84] 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 2014)*, Taipei, Taiwan, 2014.

[85] Y. Weiss. Segmentation using eigenvectors: a unifying view. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 975–982 vol.2, 1999. doi: 10.1109/ICCV.1999.790354.

[86] J. Wellhausen and M. Hoynck. Audio thumbnailing using mpeg-7 low level audio. In J. Smith, S. Panchanathan, and T. Zhang, editors, *Proceedings of the SPIE: Internet Multimedia Management Systems*, volume 5242, pages 65–73, 2003.

[87] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[88] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *CVPR*, pages 2528–2535. IEEE, 2010.

[89] R. Zhou, M. Mattavelli, and G. Zoia. Music onset detection based on resonator time frequency image. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(8):1685–1695, Nov 2008.