A Deep Learning Approach to Speech, Music and Environmental Noise Classification

Master Thesis

Felix Rothmund

Supervisors:
Univ.Prof. Dipl.-Ing. Dr.techn. Alois Sontacchi (IEM)
Dr.-Ing. Dipl.-Inf. Gerald Bauer (Harman)

Graz, March 19, 2018

Statutory declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Felix Rothmund
Graz, March 19, 2018

Abstract

The goal of this thesis is to develop an algorithm that effectively classifies audio data into the categories music, speech and environmental noise. Traditionally, audio classification tasks have been approached with standard classification procedures applied to hand-crafted descriptive features derived from the audio waveform. In recent years, inspired by their success in image classification and object recognition, deep neural nets (DNNs) have been applied to different audio classification tasks with promising results. This work assesses both traditional machine learning algorithms, as well as state-of-the-art deep learning methods for real-time audio classification. The algorithms are evaluated with audio data, coming from a far-field microphone array in different domestic and business environments. More specifically, a support vector machine (SVM) with non-linear kernel is evaluated with different descriptive audio features, as well as a convolutional neural net (CNN) applied to Mel-spectrograms. While classification accuracy is excellent for both algorithms when classifying *'clean'* audio data, the SVM performs poorly for *'real-world'* far-field microphone array recordings. An accuracy rate of over 94% is achieved using the CNN for audio clips of 1 second, providing excellent performance in real-time tests.

Zusammenfassung

Ziel dieser Arbeit ist der Entwurf und die Evaluierung eines Algorithmus zur Klassifizierung eines Audiodatenstroms in die Kategorien Musik, Sprache und Alltagsumgebungsgeräusche. Herkömmlicherweise wurden zur Audioklassifizierung konventionelle statistische Klassifikationsverfahren in Verbindung mit sogenannten Merkmalsvektoren verwendet, welche das Audiosignal mit wenigen Beschreibungsgrößen charakterisieren. Angeregt durch die erfolgreiche Anwendung in den Bereichen Bildklassifikation und Objekterkennung, wurden in den letzten Jahren sogenannte Deep Neural Nets zur Audioklassifizierung verwendet. Diese Arbeit beurteilt sowohl herkömmliche Klassifizierungsverfahren, als auch sogenannte Deep Learning Methoden für die Klassifizierung von Audiodaten in Echtzeit. Die Algorithmen werden mit Aufnahmen einer Fernfeld-Mikrofonanordnung in verschiedenen häuslichen und gewerblichen Umgebungen evaluiert. Im engeren Sinne wird eine nicht-lineare Support Vector Machine (SVM) in Verbindung mit verschiedenen Beschreibungsgrößen, und ein Convolutional Neural Net (CNN) in Verbindung mit Mel-Spektrogrammen untersucht. Während für saubere Signale beide Klassifikationsverfahren exzellente Erkennungsraten erzielen, funktioniert die SVM für Aufnahmen mit den Fernfeld-Mikrofonaufnahmen sehr schlecht. Mit dem CNN werden für Audio-Clips von 1 Sekunde Erkennungsraten von über 94% erreicht.

Contents

1 | Introduction

The goal of machine perception is to develop machines that are capable of processing and interpreting sensory input data in order to perceive and understand their surroundings like humans do.

Being one of the five traditional senses (vision, hearing, touch, taste and smell), hearing allows us to communicate with each other through vocalised speech, enjoy music and is essential for orientation in the environment. For example, an acoustic event might warn us of dangers, such as wild animals or approaching vehicles, even when our view is obstructed [1].

Starting with Bregman [2] and Brown & Cooke [3] in the early 1990s, researchers have tried to exploit the mechanics of the human auditory system for computational analysis of sound. Since then, computational auditory scene analysis (CASA) and in particular audio classification has attracted a lot of research.

With the rise of virtual assistants like *Cortana* [1.1], incorporating highly accurate speech recognition systems, people have come to expect human-level auditory perception of machines. Ideally, machines should be able to easily distinguish speech from music and background noise, locate the source of a sound and react to acoustic events in real-time [4].

For this thesis, different algorithms coming from the field of artificial intelligence are evaluated for real-time audio classification into the categories speech, music and environmental noise.

Thesis Overview
This chapter provides a detailed problem description (chapter 1.1), a general overview and summarises related research (chapter 1.2).

In chapter 2, fundamental signal characteristics are summarised for the three categories speech, music and environmental noise.

In chapter 3, different classification algorithms, as evaluated for this thesis, are described. Both traditional feature-based machine learning algorithms (chapter 3.1), as well as state-of-the-art deep learning methods (chapter 3.2) are introduced.

1.1. *Cortana* is a virtual assistant developed by *Microsoft*, recognising and processing voice-commands. $https://www.microsoft.com/en\text{-}us/cortana$ - accessed: December 2017

Chapter 4 describes the audio data used to train and validate the classification algorithms investigated.

Several feature-based audio classification algorithms are investigated. Chapter 5 describes the implemented descriptive features (chapter 5.1) and the different machine learning algorithms in more detail, including a simple Gaussian naive bayes (NB) classifier (chapter 5.2), a k-nearest neighbours (kNN) algorithm (chapter 5.3) and a support vector machine (SVM) with radial basis function (RBF) kernel (chapter 5.4).

Chapter 6 describes the deep neural nets, namely so-called convolutional neural nets (CNNs), investigated for this thesis in more detail and summarises their performance for speech, music and environmental noise classification. As described in chapter 6.8, a prototype real-time classification procedure was implemented to demonstrate the algorithm's performance in real-world environments.

Chapter 7 summarises the findings of this thesis and provides an outlook for future experiments and applications.

1.1 Problem Description

Invoke (see figure 1.1), *Harman Kardon*'s recently introduced voice-activated portable loudspeaker features a state-of-the-art beam-forming microphone array for voice commands and phone-calls.

In order to make future devices more aware of their surroundings, and thus clearing the path for new services, different audio classification algorithms are evaluated with emphasis on audio data coming from the device's far-field microphone array.

More specifically, the goal of this thesis is to develop an algorithm that effectively classifies an audio data stream into the categories music, speech and environmental noise in real-time.

While this might sound trivial, as most people can do this easily without thinking much about it, this can be a challenging task for machines and in some cases impossible even for trained listeners. This is especially the case in noisy or reverberant environments and is even more challenging for short signal segments, as needed for real-time classification.

Figure 1.1 – Harman Kardon Invoke [5]

Figure 1.2 shows the basic signal-flow for real-time classification: Audio data recorded with a microphone array is combined into one or more beam channels using a filter-and-sum beamforming technique [1.2]. From the audio data, so-called descriptive features are calculated by a feature extraction module. The classifier at the end outputs a class label (i.e. 'music', 'speech' or 'environmental noise') or an a-posteriori probability for each class.

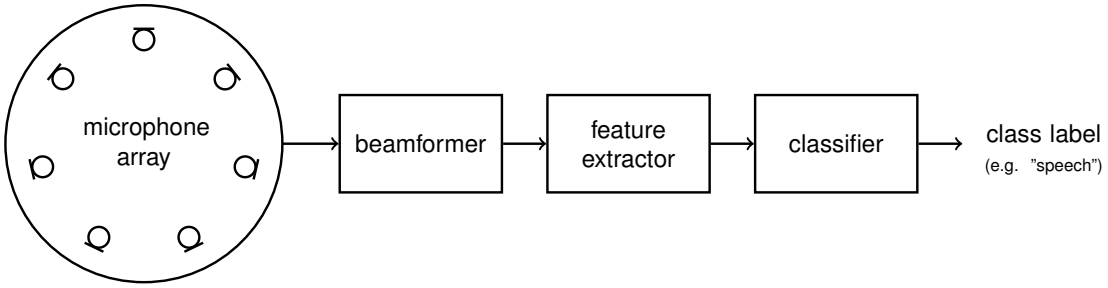


Figure 1.2 – Real-time classification of an audio data stream coming from a microphone array with subsequent beam-forming.

1.2. As the name suggests, a 'filter-and-sum' beamformer works by adding up filtered sensor data from different microphones [6].

Several classification approaches are evaluated, including traditional machine learning algorithms (chapter 5), as well as deep learning methods inspired by recent developments in computer vision (chapter 6).

In the following, an overview of related research is provided.

1.2 Related Work

Given the many potential applications in multimedia analysis, home automation and medical care, computational auditory scene analysis (CASA), and in particular audio classification has attracted a lot of research during the last decades.

Early Days
R. Clark Jones has filed a patent for a device discriminating music and speech signals as early as 1951 [7]. The proposed electric circuit was designed to be used in radio receivers to automatically silence radio signals that are predominantly speech. Classification was done monitoring rapid variations in signal energy, which occur more often in speech than music signals [8].

Feature-based Audio Classification
Starting with John Saunders in 1996 [9], audio classification has mostly been approached with descriptive audio features derived from the audio waveform, applied to different classification algorithms. As will be described in more detail in chapter 5.1, classification is usually done using clip-level features, with typical clip lengths of one to several seconds [10].

Saunders [9] successfully discriminated music and speech on broadcast radio using a multivariate Gaussian (MVG) classifier trained with features based on the zero-crossing rate (ZCR) and short-time energy (STE), claiming a classification accuracy of around 98% for audio clips of 2.4 seconds.

In 1997, Eric Scheirer and Malcolm Slaney [11] proposed a system for speech music discrimination that uses 13 different spectral, temporal and cepstral features in conjunction with a Gaussian mixture model (GMM) and k-nearest neighbours (kNN) algorithm as classifiers. They claim excellent recognition rates of 98% at clip-level (2.4 seconds).

Zhu Liu et al. [12] have classified audio clips from television broadcasts into five scene classes: news reports, weather reports, advertisement, basketball games and football games. They used 13 features based on the signal energy, fundamental frequency and spectral distribution in conjunction with a neural network classifier (one-class-in-one-network structure) with promising results.

In 1999 Zhang et al. [13] presented a hierarchical multi-expert system that segments and classifies audio data in a coarse-level and fine-level stage. On coarse-level, the audio stream is segmented and classified into the categories music, noise, silence and speech based on morphological and statistical analysis of short-term features (STE, ZCR and fundamental frequency). In the second stage, environmental sounds

are classified in more detail using a Hidden-Markov-Model (HMM) and 65 FFT-bins for classification. They report an accuracy of over 90 % at coarse-level.

Williams and Ellis [14] evaluated features derived from a hybrid connectionist-HMM-based speech recognition system for speech music discrimination, claiming an accuracy rate of 98% for voice activity detection (VAD) at clip-level (2.5s). Classification was performed by calculating means and variances separately for the speech and music training examples and performing a Gaussian likelihood ratio test.

In 2000, Khaled El Maleh et al. [15] used line spectral frequencies together with the ZCR as features and discriminated speech and music using a kNN classifier with promising results (80% at frame-level and 80-100% at clip-level (1s)).

In 2001, De Santo et al. [16] proposed a multi-expert system for audio classification into seven categories. The proposed algorithm first detects silence by simple energy thresholding. In a second stage, the non-silent signal is classified into to macro classes music and speech by a multi-layer perceptron (MLP) neural net. The two macro classes are then sub-classified further by two MLP classifiers in the third stage. They report an overall accuracy of 77% for classification in seven categories.

Lie Lu et al [17] used cascaded support vector machines (SVMs) to classify audio segments of 1s into the categories silence, music, background noise, pure speech and non-pure speech. They used the means and standard deviations for 8 mel frequency cepstral coefficients (MFCC) and several perceptual features. They report a high accuracy of around 96.5% and constitute much better performance to the SVM than other classification algorithms like kNN and GMM.

Tong Zhang and C.-C. Jay Kuo [18] investigated audio segmentation and classification for content analysis of audiovisual data. Simple descriptive audio features (energy, zero-crossings, fundamental frequency and spectral peak tracks) where analysed using a rule-based heuristic approach in order to classify audio segments into the categories silence, speech, environmental sound, pure music, song and speech with background music. They claim an accuracy of just over 90% for their efficient real-time system.

Michael C. Büchler [19] studied different classification algorithms for audio classification in hearing aids and proposed a multi-expert system in two stages. Four classes (speech, speech in noise, noise and music) were discriminated using descriptive audio features and a HMM classifier, followed by a rule-based post-processing where the output of the HMM is corrected if necessary. Büchler reports a hit rate of 91%.

In 2002, Bugatti et al. [20] trained a multi-layer perceptron (MLP) with various energy, zero-crossing and spectral features, as well as MFCCs and achieved a classification accuracy of 95% to 96% for speech music discrimination and audio segments of around 2s.

In 2003, Hadi Harb and Liming Chen [21] used statistical features derived from

Mel-spectrograms in conjunction with an MLP as a classifier for speech music discrimination. They report promising results (96% accuracy), even when training with little data only (40s of audio per class) for audio clips of 0.2 to 4s.

In 2004, Juan J. Burred and Alexander Lerch [22] classified audio data into three speech classes, 13 musical genres and background noise, using a hierarchical tree-structure of classifiers (kNN and GMM). They carried out an extensive feature selection for each branch fork in the classification tree and report an accuracy of around 95% for high-level classification in the categories speech, music and background noise.

M. Kashif Saeed Khan et al. [23] investigated MLPs and statistical metrics of a discrete wavelet transform in addition to other descriptive features for speech music discrimination. For audio clips of 3s and cross-validation, they achieved a promising classification accuracy of up to 96.6%. In 2006, M. Khan and W. Al-Khatib [10] compared MLPs and other classifiers for speech and music discrimination, reporting good performance for MLPs as well as HMMs, indicating however that HMMs require more training time. The best results were achieved using an MLP with six descriptive features (range of zero-crossings, variance of a discrete wavelet transform, RMS of low-pass signal, spectral flux, linear predictive coefficients (LPCs) and the variance of four MFCCs).

Lei Chen et al. [24] classified audio data into five categories: music, speech, environment sound, speech mixed with music and music mixed with environment sound. They compared different classifiers (SVM, kNN, naive bayes (NB) and artificial neural nets (ANN)), reporting superior performance using SVMs. For each audio segment of 1s, four audio features (ZCR, silence ratio, harmonic ratio, sub-band energy) and respective statistical representations (mean, minimum, maximum, (maximum + minimum)/2) are calculated. They achieved an overall accuracy of 78% using an SVM with Gaussian kernel, outperforming other classifiers.

In 2009, Yiizhar Lavner and Dima Ruinskiy [25] proposed an efficient speech music discriminator based on a rule-based decision-tree algorithm. Features were mean and standard deviation of STE, ZCR, sub-band energy ratio, autocorrelation coefficients, MFCCs, spectral roll-off, spectral centroid, spectral flux and spectral spread. They report music and voice detection rates of around 98%.

In 2017, M.Won et al. [26] used several time-based and spectral features with a non-linear SVM to classify audio data into the categories speech, music, noise and speech over music in noisy in-vehicle environments. As different acoustical environments significantly degrade classification performance, they developed an algorithm that adapts based on the driving environment in order to achieve high accuracy rates.

Recent developments and Convolutional Neural Nets

Inspired by their tremendous success in image classification [27], researchers have applied so-called convolutional neural nets (CNNs) to audio classification.

In 2016, Justin Salamon and Juan P. Bello [28] proposed a CNN architecture for

environmental sound classification into 10 classes. To overcome the lack of training data, they propose the use of audio data augmentation. They generate additional audio data using time-scale modification (time- and pitch-shifting), dynamic range compression techniques and additive background noise. They report state-of-the-art results for multi-class environmental sound classification.

Lukic et al. [29] used CNNs for speaker identification and clustering. More specifically, they discriminated many different speakers by analysing simple spectrograms, and studied the optimal model structure for the CNN. They report results comparable to state-of-the-art systems without the need for hand-crafted descriptive features.

In 2017, Naoya Takahashi et al. [30] proposed a CNN for audio classification, operating on spectrograms several seconds long. They report significant performance improvements on action recognition tasks for audio-visual data compared to visual features alone and visual features with MFCC-feature based approaches.

In 2017, Gemmeke et al. [31] published *Audio Set*, an extensive ontology for audio event description and a massive human-labeled audio dataset from YouTube-videos. Shawn Hershey et al. [32] used the data for large-scale audio classification using different neural net topologies, finding that CNNs outperform other topologies and feature-based approaches.

Summary
Excellent classification accuracy is reported for a wide variety of algorithms. Traditionally, standard statistical classification procedures are applied to hand-crafted descriptive features derived from the audio waveform.

Classifiers include Gaussian mixture models (GMM), naive bayes (NB) and k-nearest neighbours (kNN) classifiers, non-linear support vector machines (SVM) as well as multi-layer perceptrons (MLP).

While many authors report excellent classification accuracy, it is hard to compare the proposed algorithms. For one, they use different datasets and clip lengths (time resolution) and maybe even more importantly, classify audio into different categories and sub-categories.

For most research, 'clean' low-noise broadcast audio was used to train and evaluate the algorithms, where the waveform is not heavily degraded by room impulse responses, additive noise and other signal distortions. As reported by Won [26], feature-based classifiers are prone to noise and changes of acoustic parameters. So the performance is expected to be worse for far-field audio data in noisy environments, as investigated for this thesis.

More recent research successfully applies CNNs, deep neural nets borrowed from the field of computer vision [33], to audio classification tasks, eliminating the process of feature extraction and selection, while reporting similar, if not better performance. CNNs have been proven to be noise-robust in speech recognition tasks [34], making them a promising contender for audio classification in varying and noisy acoustic environments.

2 | Signal Characteristics

In order to find meaningful descriptors, capable of discriminating speech, music and environmental noise signals, it makes sense to study their characteristics first.

In the following, fundamental signal characteristics of speech, music and environmental noise signals will be described.

2.1 Speech

Speech describes the ability of humans to communicate with each other through vocalised sounds produced by specialised vocal organs.

> **speech** [ˈspiːt͡ʃ] *noun* • The expression of or the ability to express thoughts and feelings by articulate sounds [35].

Speech signal characteristics are well-studied and have been successfully exploited in various applications, such as speech-coding or automatic speech recognition (ASR) [36].

Human speech is produced by the vocal organs shown in figure 2.1. The lungs provide energy in the form of compressed air, which then passes through the larynx, where the airflow can be altered in different ways. When the vocal chords are under tension, the airflow will cause them to vibrate. This causes the periodic release of air, which will result in voiced sounds. If the vocal chords are relaxed, this results in a turbulent airflow, resulting in wide-band noise or unvoiced sounds. The excitation signal (either pulse train or wide-band noise) travels through the pharynx, nasal and oral cavities which act as a resonant acoustic filter [37].

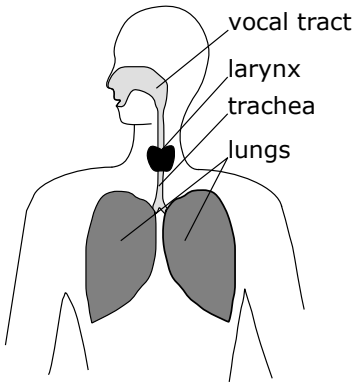


Figure 2.1 – The human organs of speech production [36].

The nature of speech production can be approximated with the so called source-filter model, as depicted in figure 2.2. The vocal chords / larynx are modelled as either a generator of wide-band noise or a pulse-train at fundamental frequency f_0. The vocal tract is modeled by a resonant predictive filter [36].

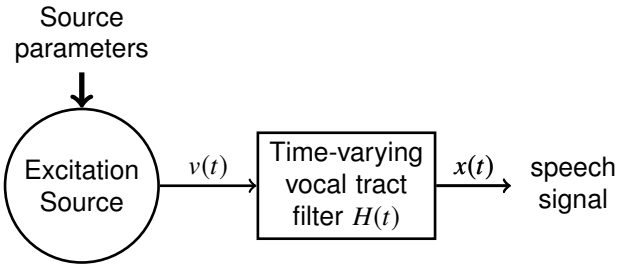


Figure 2.2 – Source-filter model: The excitation signal $v(t)$ (i.e. pulse train or wide-band noise) is generated by the excitation source and is characterised by its source parameters. The speech signal $x(t)$ is the excitation signal filtered by a time-varying filter $H(t)$, modeling the vocal tract [36].

Voiced and unvoiced speech
In essence, a speech signal is a sequence of syllables and pauses. On average, English speech consists of around 4 syllables per second [38]. Generally spoken, syllables consist of a *voiced* syllable nucleus (vowel) and optional onset and coda, which can be *unvoiced* (consonants) or *voiced* (semi-vowels and voiced consonants).

Fundamental Frequency and Prosody
As mentioned above, a speech signal can be either voiced or unvoiced. Voiced speech segments are characterised by their pitch and the energy distribution among the lower order harmonics. The period of the vocal chord pulse train determines the fundamental frequency f_0 of a speech segment. For men, the average fundamental frequency is around $f_0 = 120$ Hz, for women around $f_0 = 200$ Hz with a standard deviation of around ± 20Hz [39].

The fundamental frequency or pitch is not stationary but varies to emphasise certain speech segments (e.g. to signify a question). This fluctuation of pitch is part of what is called *prosody* and can be observed when looking at the lower graph in figure 2.3.

Figure 2.3 shows the audio waveform of a speech signal $x(t)$ and the corresponding magnitude spectrogram $|X[k,t]|_{\text{dB}}$ in dB. As can be seen, voiced and unvoiced speech segments can be to some extend discriminated visually, both in the waveform (top) and spectrogram [2.1] (bottom).

Voiced segments like vowels and voiced consonants are of greater signal energy, indicated by greater amplitudes in the waveform (top). Amplitudes vary quickly with segments of silence (amplitude close to zero), unvoiced consonants with low

2.1. The spectrogram describes the distribution of signal energy among time and frequency and is described in more detail in chapter 5.1

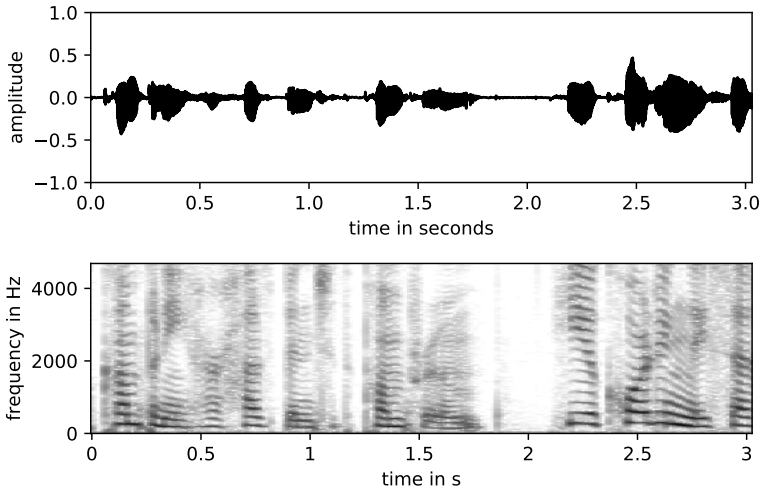


Figure 2.3 – Speech signal of female speaker. (top) Audio waveform for 3s of audio and (bottom) corresponding magnitude spectrogram in dB.

amplitudes or short energy bursts (e.g. plosives [2.2]) and voiced segments with larger amplitudes. The graph shows 1 second of audio, containing 4 syllables of speech.

In the spectrogram, voiced speech segments are characterised by horizontal lines in the lower frequency range, representing the fundamental frequency f_0 and lower order harmonics (multiples of f_0). Unvoiced segments are characterised by their bandwidth and distribution of energy along frequency.

Note that the described signal characteristics apply to normal human speech of male and female adults. For special modes of phonation, like whispering or screaming, characteristics may differ and are not investigated. Also, only speech signals with one single speaker are evaluated. Mixtures of multiple simultaneous speakers may be interesting for future work. As will be described in more detail in chapter 4, both male and female speech is examined.

2.2. Plosives, also known as stop consonants describe articulate sounds produced by blocking and releasing compressed air in the vocal tract. e.g. [p],[t],[k], ... [40]

2.2 Music

Generally spoken, the term *music* describes all sounds intentionally produced by humans using musical instruments or their voice, intended to invoke pleasure or satisfaction.

> **music** [ˈmjuːzɪk] *noun* • Vocal or instrumental sounds (or both) combined in such a way as to produce beauty of form, harmony, and expression of emotion [35].

Due to its diverse nature, music cannot be parameterised as well as speech. However, two key assumptions can be made for what is referred to as western music: Music is organised temporally and harmonically, i.e. in time and frequency.

Figure 2.4 shows an excerpt of common staff notation, which graphically visualises the organisation of a musical piece in time and frequency.

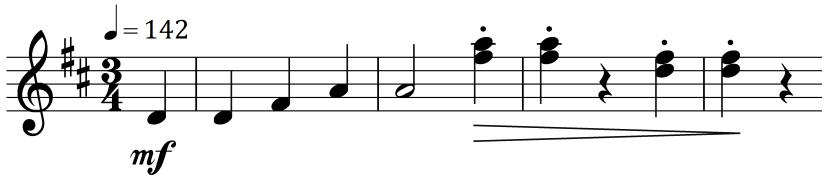


Figure 2.4 – Simplified musical notation of the main theme of 'The Blue Danube' by Johann Strauss [41].

The pitch of a musical note is denoted by its vertical placement on or between the staff lines. In tonal music, the placement of pitches determines the harmonic tonality or harmony of a piece. The horizontal placement of musical notes denotes the temporal organisation or rhythm of a piece. [41].

Most music is organised around an underlying pulse or beat. The tempo of a musical piece is defined by the number of beats per minute (bpm) (e.g. 142 bpm for the shown walz) and is typically between 40 and 200 bpm [42].

Besides the temporal and harmonic organisation, music is characterised by its temporal evolution of loudness or dynamic (as denoted by the annotation *mezzoforte* (***mf***) and the so called 'hairpin' below the staff in figure 2.4) and timbre.

Fundamental Frequency and Tonality

The perceived pitch of a musical tone is determined by the fundamental frequency f_0. In contrast to speech signals, where the fundamental frequency ranges from around 50Hz to 250Hz, the range of f_0 is a lot greater in music signals. For example, the lowest musical note on a typical piano with 88 keys (A0) is pitched at around 27.5 Hz. The highest note (C8) has a fundamental frequency of around 4,186 Hz [43].

Figure 2.5 shows the audio waveform (top) and corresponding spectrogram (bottom) of a music segment. Compared to the speech signal (figure 2.3), signal energy fluctuates less rapidly over time and frequency.

Generally, in music signals, the fundamental frequency and harmonic structure are expected to be stationary for longer segments compared to speech signals [44]. This can be observed when looking at the spectrogram of the music segment shown in figure 2.5 (bottom).

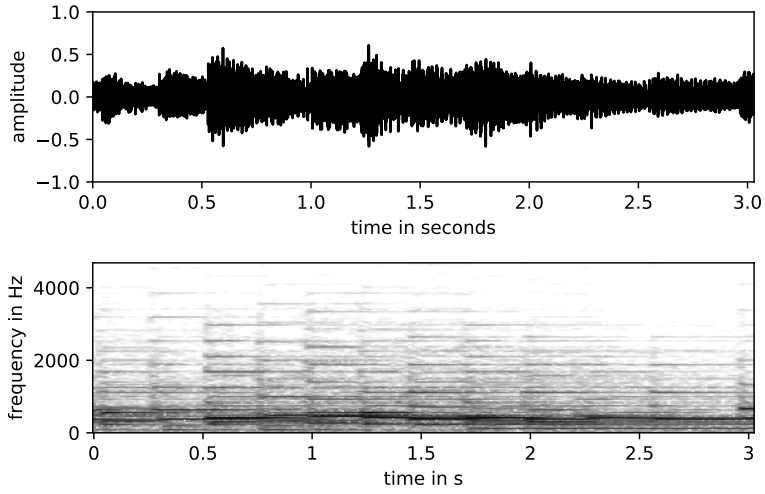


Figure 2.5 – Music signal (excerpt of piano concert). Audio waveform (top) and logarithmic power spectrogram (bottom). Straight horizontal lines in the spectrogram are characteristic for tonal music signals.

Timbre and Perceptual attributes

Besides its pitch and loudness, a musical tone is characterised by the distribution of energy among its harmonics, i.e. the shape of the spectrum and its development over time. In psychoacoustics, this is called *timbre* of a musical sound. The timbre of a sound can be described subjectively or objectively by analysing the audio waveform. Perceptual qualities of music signals have been extensively studied in the field of psychoacoustics and are exploited in various descriptive audio features, as will be described in chapter 5.1.

In summary, music is characterised by the attributes *pitch*, *loudness*, *timbre*, *rhythm* and *harmony*.

For this thesis, music recordings of various genres are evaluated, as will be described in more detail in chapter 4.

2.3 Environmental Noise

Most dictionaries describe the term *noise* as an unwanted or disturbing sound. For this thesis however, *environmental noise* describes any sound that is neither music nor speech. This includes various sounds that occur in different acoustic environments, such as domestic areas and outdoor locations, as well as business environments.

> **noise** [ˈnɔɪz] *noun* • A sound, especially one that is loud or unpleasant or that causes disturbance [35].

The nature of environmental noise sounds is very diverse and cannot be generalised satisfyingly. They include pitched sounds (e.g. electrical appliances or power tools), as well as unpitched, wide-band noise like wind or street noise.

Figure 2.6 demonstrates the diverse characteristics of noise signals.

The top graph (a) shows a stationary wide-band noise signal and the corresponding spectrogram. As can be observed, signal energy is stationary for the whole clip of three seconds and evenly spread along time and frequency.

The middle graph (b) shows an audio recording of a door being shut, resembling a transient noise event. Contrary to the wide-band noise signal (a), the amplitude rapidly fluctuates (at around 1.2s and 2.25s). The transient noise event is characterised by vertical lines in the spectrogram.

The lower graph (c) shows a recording of a power tool, resembling an environmental noise sound with tonal energy content. The pitched noise sound is characterised by its fundamental frequency and its evolution over time, as well as the distribution of signal energy among the lower order harmonics.

For this thesis, sounds from various sources and environments were evaluated. Chapter 4 describes the composition of the used data in more detail.

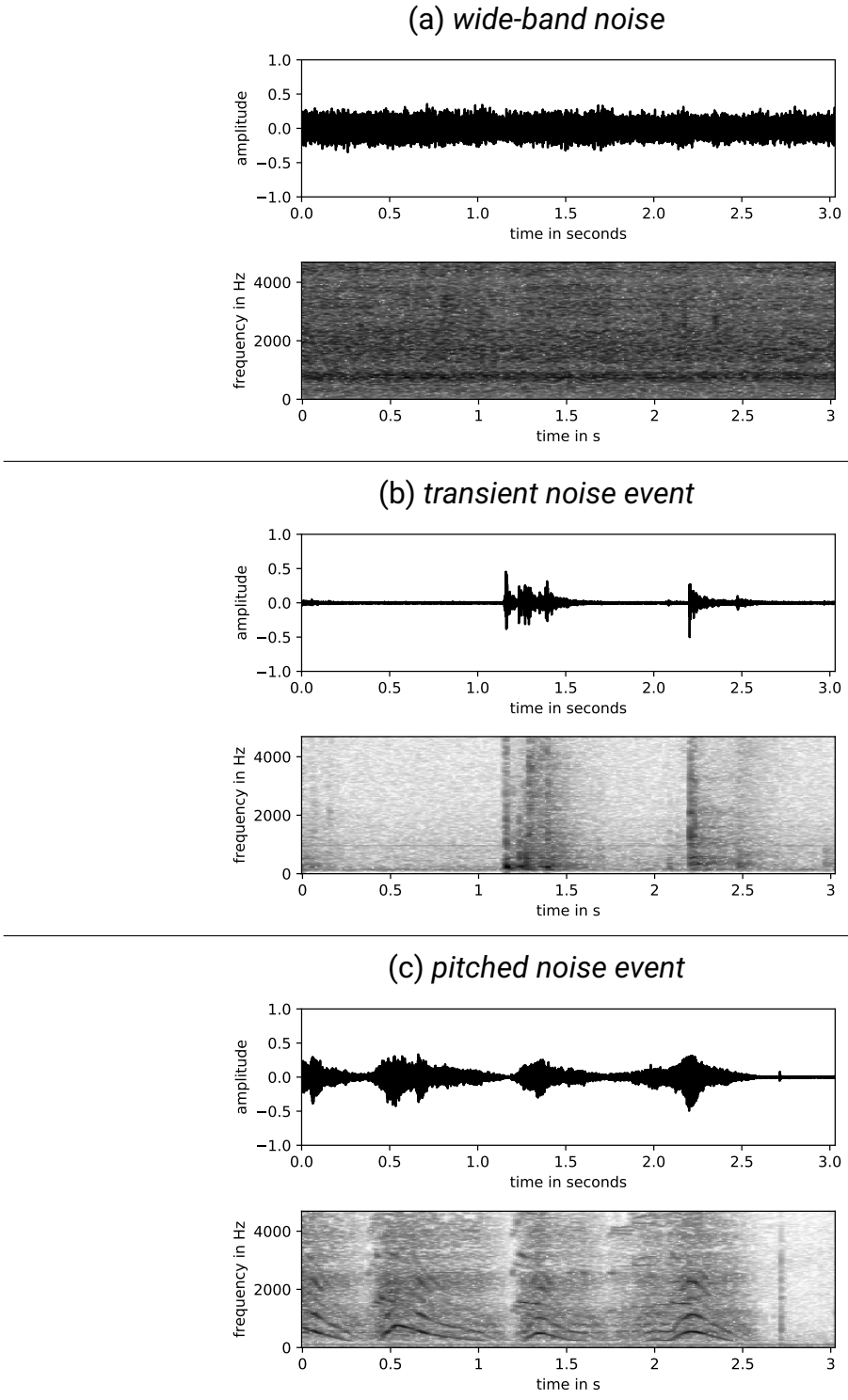


Figure 2.6 – Noise signals, plots show the audio waveform and spectrogram for (a) wide-band noise, (b) transient noise event and (c) a pitched noise sound.

3 | Classification Algorithms

The goal of machine learning (ML) is to estimate complex (non-linear) functions, that cannot be solved easily using rule-based hand-crafted systems. Typical tasks for ML algorithms are classification, regression, transcription, translation, anomaly detection and de-noising.

The term learning describes the process of optimising an algorithm's parameters based on experience (i.e. previous data samples) in order to maximise its performance. Any algorithm that learns from so-called training data can thus be called a ML algorithm [45, p.95].

This chapter provides a short introduction to the basics of ML needed for this thesis, focusing on the classification task.

The Classification Task
In classification tasks, a ML algorithm aims to associate an observation $\mathbf{x}$ to one of N_c classes, based on experience gained from statistical analysis of previous data points during the so-called training process.

The algorithm usually solves a function $f : \mathbb{R}^N \rightarrow \{1, 2, \ldots, N_c\}$, aiming to associate an observation $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ with a predicted class c:

$$c = f(\mathbf{x}) \tag{3.1}$$

Other variants output an a-posteriori probability distribution, yielding an estimated probability $\tilde{\mathbb{P}}(c)$ for observation $\mathbf{x}$ being of class c [45, p.97].

$$\tilde{\mathbf{y}} = \begin{bmatrix} \tilde{\mathbb{P}}(1) \\ \tilde{\mathbb{P}}(2) \\ \vdots \\ \tilde{\mathbb{P}}(N_c) \end{bmatrix} = f(\mathbf{x}) \tag{3.2}$$

Supervised and Unsupervised Training
ML algorithms follow two fundamental training principles. Supervised training methods aim to associate an input with a reference output (e.g. class label c), provided by a (human) supervisor. Unsupervised training methods on the other hand, learn useful properties from training data without reference data [45, p.102].

For supervised ML algorithms, the training dataset $\mathbf{X}$ consists of O observations, each with N attributes and the corresponding targets $\mathbf{y}$ (in classification task, $\mathbf{y}$ contains class labels).

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \vdots \\ \mathbf{x}^{(O)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_N^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_N^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(O)} & x_2^{(O)} & \dots & x_N^{(O)} \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(O)} \end{bmatrix}. \tag{3.3}$$

In classic ML algorithms, attributes resemble so-called descriptive features, derived from the raw input data in order to obtain a more compact input representation. Descriptive audio features evaluated for this thesis will be described in more detail in chapter 5.1.

Overfitting and Underfitting

Ideally, a ML algorithm should perform well on new, previously unknown input data. In other words, the algorithm should represent a generalised model that works on a wide variety of input data.

Training a ML algorithm usually involves minimising the so-called training error by adapting the algorithm's parameters. The training error is obtained by comparing the algorithm's output with the reference output (i.e. class label). To evaluate an algorithm's ability to generalise, the so-called test error is derived from separate test data. Samples in the test dataset are not be included in the training data and vice-versa. Test data should resemble input data, that is expected to occur in the algorithm's final application.

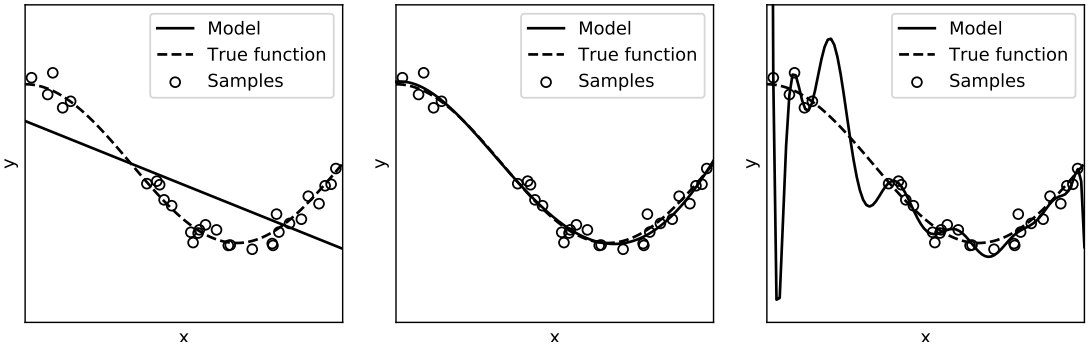
If a ML algorithm is too simple and thus not capable of achieving a low training error, the algorithm will perform poorly on test data. The algorithm's underlying statistical model fails to capture the training data's trend. This is called *underfitting*.

Overfitting on the other hand occurs, when the training error is very small, but the gap between the training and test error is too large. This indicates that the underlying model is too complex, reacting to noise in the training data rather than general trends [45, p.107-112].

Figure 3.1 shows three different estimated models with rising complexity, aiming to describe a datasets underlying trend. As can be seen, the under- and overfit models (a) and (c) fail to describe the samples' underlying function.

In general, good generalisation performance is achieved, when the capacity / complexity of the classification function is matched to the amount of available training data [46] and the offset between the error for train and test data is small.

Figure 3.2 shows how increasing the capacity of a model will decrease the test error until the optimal capacity is reached. Beyond the optimal capacity the model will overfit the training data leading to poor generalisation performance.



(a) Underfit model (b) Accurate model (c) Overfit model

Figure 3.1 – Training data points fitted with three different models, with rising complexity. If the model is too simple (a), the model cannot describe the underlying trend. If it is too complex (c), overfitting will occur [47].

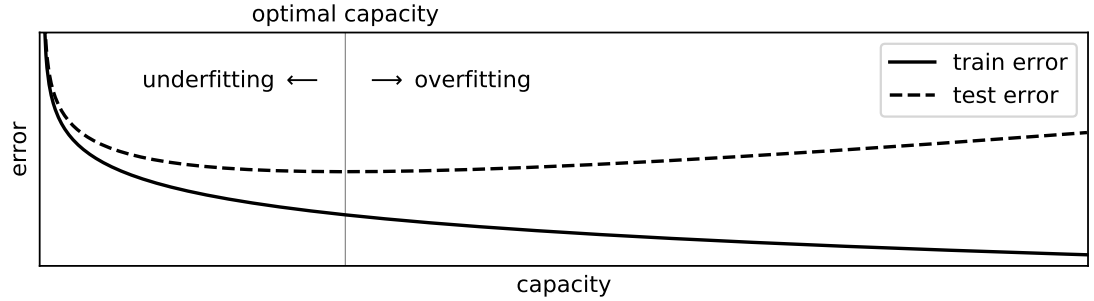


Figure 3.2 – The capacity of a model should match the amount of training data. If the model is too complex (high capacity) the model will overfit. If the capacity is too low, the model will underfit. [45, p.113]

3.1 Feature-based Classifiers

Feature-based classifiers have been proven useful for classification of broadcast audio [9, 11], as described in chapter 1.2. For this thesis, a simple Gaussian naive bayes (NB) classifier, a k-nearest neighbours (kNN) algorithm and a support vector machine (SVM) with Gaussian kernel are evaluated in conjunction with several descriptive features (see chapter 5.1).

In the sections below, the underlying mechanics of the classification algorithms will be explained.

3.1.1 Naive Bayes Classifier

The naive bayes (NB) classifier is a simple probabilistic classifier, applying Bayes' law of conditional probabilities. It is used as a baseline classifier, to compare other more sophisticated algorithms against.

The classifier is *naive* in that it assumes independence between attributes. Even though this is almost never the case in real life, the classifier still works well for a wide variety of classification tasks, as long as the attributes are note too strongly correlated [48].

Bayes' Law
Bayes' law (also Bayes' theorem or Bayes' rule) describes conditional probabilities mathematically. Given the probabilities $\mathbb{P}(A)$ and $\mathbb{P}(B)$, observing the events A and B, the probability that event A occurs, given that B is true, is given as

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A) \cdot \mathbb{P}(B|A)}{\mathbb{P}(B)}, \tag{3.4}$$

with $\mathbb{P}(B) \neq 0$, assuming that A and B are independent [45, p.68].

A NB classifier aims to predict the probability $\mathbb{P}(c|\mathbf{x})$ for an observation $\mathbf{x}$ being of class c.

Multi-class discrimination
For multi-class discrimination, the probability for observation $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ being of class c is computed as

$$\mathbb{P}(c|\mathbf{x}) \propto \mathbb{P}(c) \prod_{i=1}^{N} \mathbb{P}(x_i|c), \tag{3.5}$$

where x_i is one of N observed attributes or features for observation $\mathbf{x}$. $\mathbb{P}(c)$ describes the a-priori probability of class c. When assuming a-priori probabilities to be equal

for all classes $c = 1, 2, \ldots, N_c$, the term can be simplified to

$$\mathbb{P}(c|\mathbf{x}) \propto \frac{1}{N_c} \prod_{i=1}^{N} \mathbb{P}(x_i|c), \tag{3.6}$$

In classification tasks, the goal is to find the most probable class c for an observation $\mathbf{x}$, or in other words finding the maximum a-posterori (MAP) class c_{MAP}:

$$c_{\text{MAP}} = \underset{c}{\operatorname{argmax}}\, \tilde{\mathbb{P}}(c|\mathbf{x}) = \underset{c}{\operatorname{argmax}} \prod_{i=1}^{N} \tilde{\mathbb{P}}(x_i|c). \tag{3.7}$$

where $\tilde{\mathbb{P}}(x_i|c)$ denotes estimated probabilities learned during training [49, p.258]. To prevent underflow in the digital domain, instead of multiplying probabilities, their logarithms are summed, as follows.

$$c_{\text{MAP}} = \underset{c}{\operatorname{argmax}}\, \tilde{\mathbb{P}}(c|\mathbf{x}) = \underset{c}{\operatorname{argmax}} \sum_{i=1}^{N} \log\left(\tilde{\mathbb{P}}(x_i|c)\right). \tag{3.8}$$

Gaussian Naive Bayes Algorithm

The *Gaussian* NB algorithm assumes, that the likelihood of each attribute x_i is distributed according to the Gaussian function

$$\mathbb{P}(x_i|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(x_i-\mu_c)^2}{2\sigma_c^2}}, \tag{3.9}$$

where σ_c and μ_c denote the attribute's standard deviation and mean derived from the training data [50].

3.1.2 k-Nearest Neighbours Classifier

The k-nearest neighbours (kNN) algorithm is a classifier that estimates the class membership of previously unknown data samples based on the class membership of the k closest samples within the training data.

The kNN algorithm is a so-called instance-based learning algorithm, meaning that the training process only consists of storing the training dataset $\mathbf{X}$ (also called lazy learning) [51].

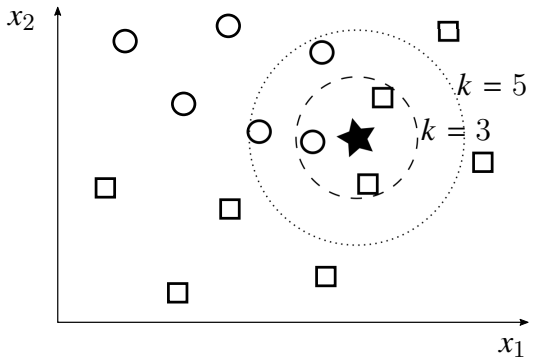


Figure 3.3 – An unknown sample (star) is classified by evaluating the class membership of the k nearest neighbours (kNN) in a two-dimensional feature space. For $k = 3$ two neighbours are of class 'square' and one of class 'circle', resulting in a predicted class 'square'. For $k = 5$, two neighbours are of class 'square' and three of class 'circle', resulting in a predicted class 'circle'.

Figure 3.3 demonstrates how the algorithm works: The k closest data points $\mathbf{x}^{(i)}$, with $i = 1, 2, \ldots, k$ within the training data $\mathbf{X}$ are evaluated for their class membership. Usually the so-called Euclidean distance is used to find the nearest neighbours, corresponding to a straight line between the unknown feature vector $\mathbf{x}$ and the training data sample $\mathbf{x}^{(i)}$. The unknown sample $\mathbf{x}$ is predicted to be of the class which occurs the most among the k nearest neighbours [45, p.141].

As can be seen, the choice of k severely affects the prediction capabilities. Generally spoken, a large k increases the analysis radius around the input $\mathbf{x}$, making the implicit decision boundary smoother and in consequence reducing the effects of local noise [51]. If k is too small, overfitting might occur, if it is too large, the algorithm might underfit the training data.

3.1.3 Support Vector Machine (SVM)

The support vector machine (SVM), is a popular algorithm for binary classification tasks, due to its relatively low computational cost during prediction and ability to find well-performing solutions with relatively little training data.

SVMs, originally introduced as maximum margin classifiers [46] separate data samples in the N-dimensional parameter-space $\mathbb{R}^N$ by finding a so-called hyperplane, that maximises the margin M between data samples belonging to two separate classes.

Figure 3.4 shows some data belonging to the two classes 'square' and 'circle'. For two-dimensional data, the hyperplane is a straight line, separating the two classes.

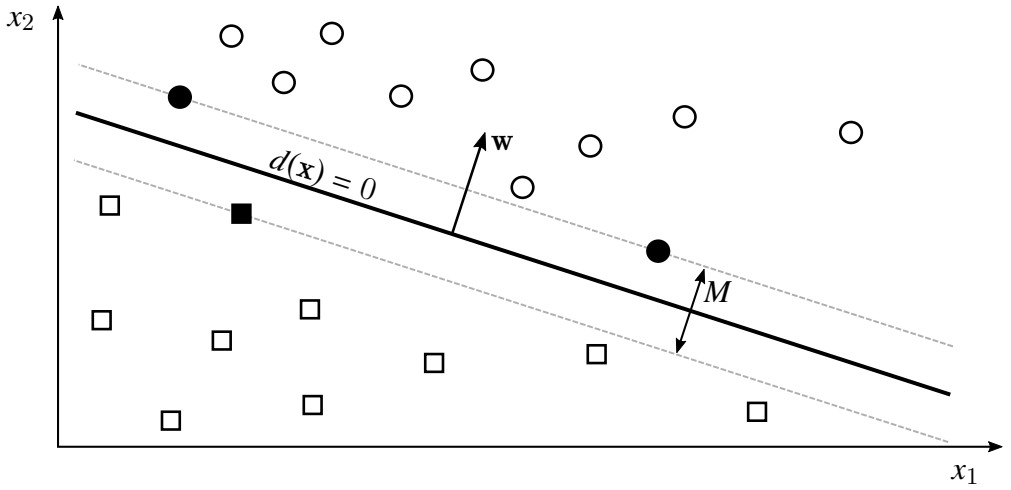


Figure 3.4 – Two linearly separable classes 'square' and 'circle' and the maximum margin hyperplane defined by the support vectors, visualised as filled circles and squares respectively.

The algorithms underlying decision function $d(\mathbf{x})$ is defined as the dot product of a weight vector $\mathbf{w}$ and the input vector $\mathbf{x}$

$$d(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{3.10}$$

Essentially, data points on one side of the hyperplane ($d > 0$) are associated with one class ($y = +1$), data points on the other side ($d < 0$) with the other class ($y = -1$). The weight vector $\mathbf{w}$ is normal to the hyperplane separating the two classes, defining the orientation of the hyperplane in the multidimensional space $\mathbb{R}^N$. The bias b defines its shift from the origin $\frac{b}{||\mathbf{w}||}$.

In order to maximise the margin M we define the constraints

$$\text{for } y^{(i)} = +1: \qquad = \mathbf{w}^T\mathbf{x}^{(i)} + b \geq \quad a, \tag{3.11}$$

$$\text{for } y^{(i)} = -1: \qquad = \mathbf{w}^T\mathbf{x}^{(i)} + b \leq -a. \tag{3.12}$$

for all training examples $\{\mathbf{x}^{(i)}, y^{(i)}\}$, or simpler

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq a \quad \forall\, i, \tag{3.13}$$

with an arbitrary scaling variable a. From this we can derive the function for the margin

$$M = \frac{2a}{||\mathbf{w}||}, \tag{3.14}$$

Maximising the margin M is thus equivalent to minimising the norm $||\mathbf{w}||^2$

$$M = \arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 \quad \text{with constraint} \quad y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 \quad \forall\, i = 1, \ldots, O. \tag{3.15}$$

In other words, the weight vector $\mathbf{w}$ is a linear combination of the training examples $\mathbf{x}^{(i)}$ for all observations $i = 1, 2, \ldots O$ in the training dataset $\mathbf{X}$:

$$d(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{O} \alpha_i \mathbf{x}^T\mathbf{x}^{(i)} + b \tag{3.16}$$

In practice, it is sufficient to store the so-called support vectors $\mathbf{x}^{(i)}$ with non-zero weights $\alpha_i \neq 0$, as depicted as filled circles and squares in figure 3.4 [45, 52].

Kernel Trick

Due to the linearity of the decision boundary, standard SVMs can only classify linearly separable data. However, by applying a non-linear function $\Phi : \mathbb{R}^{N_1} \rightarrow \mathbb{R}^{N_2}$ to the input $\mathbf{x}$, transforming it into a higher-order feature space $\mathbb{R}^{N_2}$ ($N_2 > N_1$), non-linearly spaced data can be made separable using a linear decision function (as demonstrated in figure 3.5).

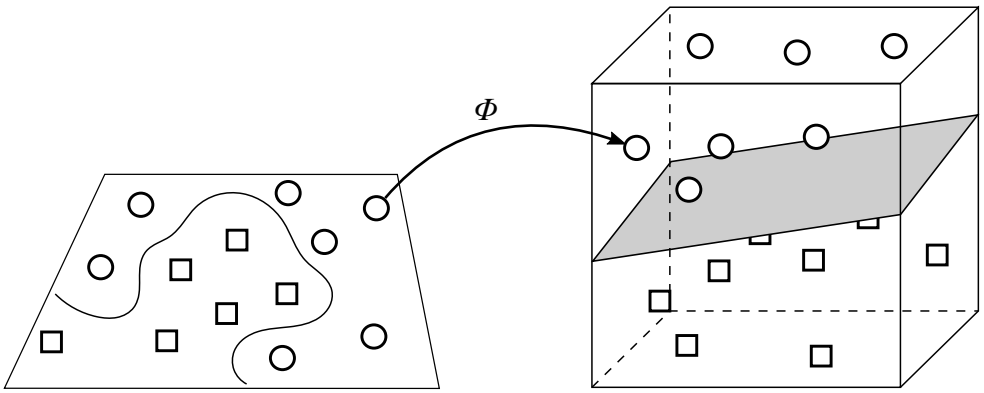


Figure 3.5 – Applying a non-linear transformation Φ to the input feature space allows to separate non-linearly spaced data with a linear hyperplane [53].

Actually applying the transform $\Phi(\mathbf{x})$ to the input data is computationally demanding, and sometimes even impossible (i.e. if N_2 is infinite). The so-called kernel

trick simplifies the transformation to a higher-order feature space, by simple kernel evaluations. The decision function (see formula 3.16) can be rewritten as

$$d(\mathbf{x}) = \sum_i \alpha_i \, K(\mathbf{x}, \mathbf{x}^{(i)}) + b \tag{3.17}$$

replacing the dot product $\mathbf{x}^T\mathbf{x}^{(i)}$ with the (non-linear) kernel function

$$K(\mathbf{x}, \hat{\mathbf{x}}) = \sum_i \varphi_i(\mathbf{x})\varphi_i(\hat{\mathbf{x}}). \tag{3.18}$$

Gaussian Kernel
The most commonly used kernel is the Gaussian or radial basis function (RBF) kernel

$$K(\mathbf{x}, \hat{\mathbf{x}}) = e^{-\gamma \, ||\mathbf{x}-\hat{\mathbf{x}}||^2}, \qquad \gamma = \frac{1}{2\sigma^2}, \tag{3.19}$$

resembling the Gaussian standard normal density. $||\mathbf{x}-\hat{\mathbf{x}}||^2$ is the squared Euclidean distance between $\mathbf{x}$ and $\hat{\mathbf{x}}$, the open parameter γ controls the variance σ^2 of the Gaussian RBF.

When an unknown sample $\mathbf{x}$ is located near a training sample $\mathbf{x}^{(i)}$, the RBF kernel function will yield a strong response, indicating that they are similar, putting a large weight on the associated training label [45, p.140].

A small value for γ, i.e. a large variance σ^2 will expand the influence of the support vector $\mathbf{x}^{(i)}$ on the classification result for $\mathbf{x}$. A large value for γ on the other hand means that only support vectors very close to $\mathbf{x}$ will have an influence on the classification result.

SVM with Soft Margin
The standard SVM aims to find an optimal decision boundary to separate all data samples in the training set. However, this means that the classifier might overfit the data (or in case of a linear kernel might not be possible). To alleviate this problem, a so-called *soft margin* decision boundary is used [54].

The optimisation problem (equation 3.15) then extends to

$$M = \arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \cdot \sum_i \xi_i \qquad \text{with constraint} \quad y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i, \tag{3.20}$$

penalty parameter C and slack variables ξ_i [52, 54, 55].

Essentially, the penalty parameter C controls the number of support vectors that are either within the margin or on the *'wrong'* side of the decision function ($\xi_k \neq 0$). If C is small, the decision boundary will be smoother, making the model less prone to overfitting. A large penalty C will lead to a more complex decision boundary, describing the training data more accurately. The goal is to find a parameter C that explains the actual trend in the training data without being too complex (overfitting) or too simple (underfitting).

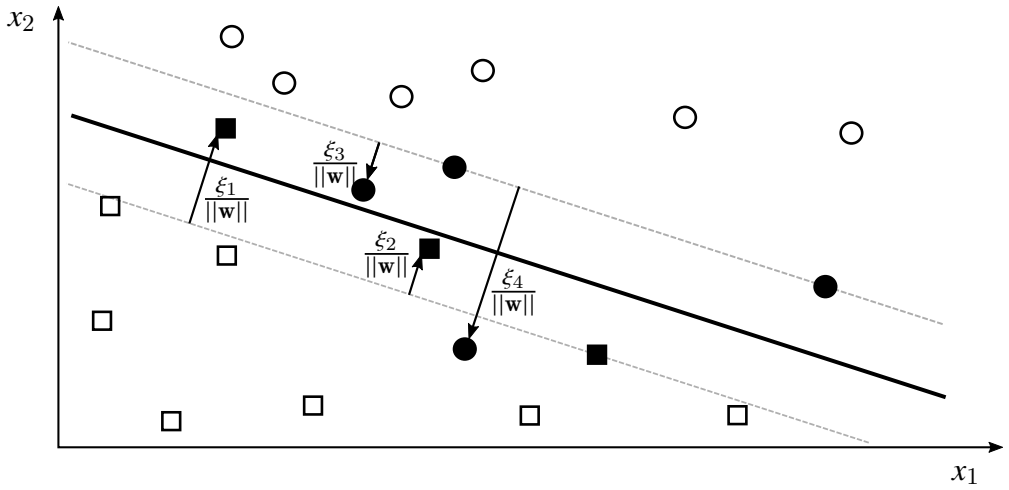


Figure 3.6 – Using a soft-margin can reduce overfitting, by allowing some support-vectors to be within the margin or even on the 'wrong' side of the decision boundary. [52]

Multi-class discrimination

For multi-class classification tasks ($N_c > 2$), several SVMs can be combined to form a multi-class classifier.

The most efficient and straight-forward implementation is the *One-vs-Rest* method. A classifier is trained separately for each class c, with samples of class c being positive ($y = 1$) and the rest being negative ($y = -1$). The maximum output score (e.g. distance from decision boundary) then defines the output class

$$c = \underset{c}{\text{argmax}}\ f_c(\mathbf{x}). \tag{3.21}$$

Another, more computationally demanding method is the 'One-vs-One' method. $\frac{N_c(N_c-1)}{2}$ binary classifiers are trained, one for each unique pair of classes $\{c_i, c_j\}$. All binary output predictions are then combined by counting the number of positive predictions for each class.

$$c = \underset{c}{\text{argmax}}\ g_c(\mathbf{x}) \quad \text{with} \quad g_c(\mathbf{x}) = \sum_{k=1}^{\frac{N_c(N_c-1)}{2}} \delta(c, f_k(\mathbf{x})) \tag{3.22}$$

with the Kronecker delta $\delta(a, b)$ given as

$$\delta(a,b) = \begin{cases} 1 & \text{for } a = b, \\ 0 & \text{otherwise .} \end{cases} \tag{3.23}$$

3.2 Deep Learning Algorithms

The term deep learning describes ML algorithms incorporating artificial neural networks with three or more layers of artificial neurons. Deep learning algorithms include both supervised and unsupervised training algorithms and are used for clustering and classification tasks, as well as regression tasks. By adding more layers, a deep neural network can estimate functions of increasing complexity.

Especially so-called convolutional neural nets (CNN) have been tremendously successful in image classification and object recognition tasks, outperforming all previous approaches [27].

In contrast to traditional ML algorithms, deep neural nets can extract features from raw input data by non-linear transformations, outperforming hand-made descriptive features [45].

3.2.1 Artificial Neural Net (ANN)

Artificial neural networks (ANNs) are an approach to model the learning capabilities of the central nervous system of animals. Also being referred to as connectionist systems they are inspired by biological neural networks, which consist of interconnected neurons.

Figure 3.7 shows the biological model of a neuron. It consists of the cell body and the axon.
The branches stemming from the cell body form the dendritic tree which collects electrochemical signals from neighbouring neurons.
If there is enough stimulation collected by the dendrites, an electrical spike will move along the axon, passing the electrical signal on to other neurons like a transmission line [57].

This behaviour is modelled mathematically by interconnected processing units called artificial neurons.

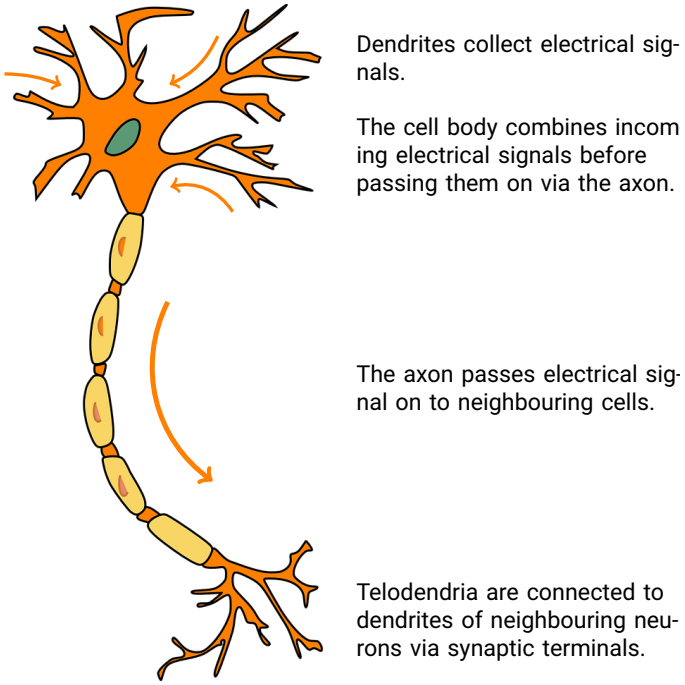


Figure 3.7 – Biological model of a neuron consisting of the cell body and the axon [56].

Artificial Neurons

An artificial neuron combines the input values $\{x_1, x_2, \ldots, x_N\}$ by linear combination (see figure 3.8). Each input x_i is weighted by multiplying with a corresponding weight w_i. The sum of the weighted inputs and the bias b is called activation a, which is then transformed using a non-linear activation function $\sigma(a)$, yielding the neuron's output state y.

$$a = \sum_{i=1}^{N} w_i \cdot x_i + b \qquad\qquad y = \sigma(a), \tag{3.24}$$

which can be rewritten as a dot product of a weight vector $\mathbf{w}$ and input vector $\mathbf{x}$

$$y = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right). \tag{3.25}$$

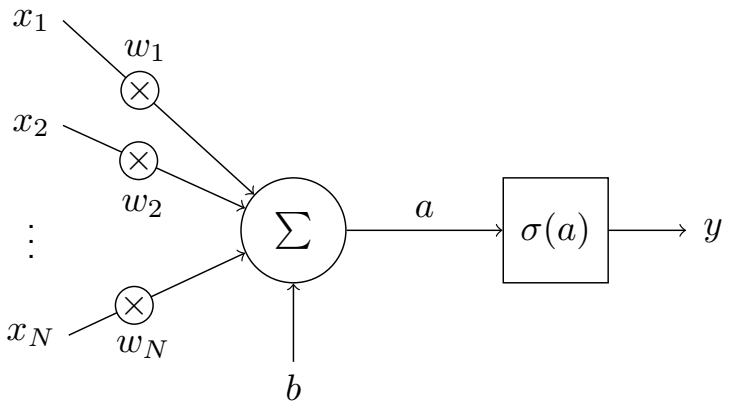
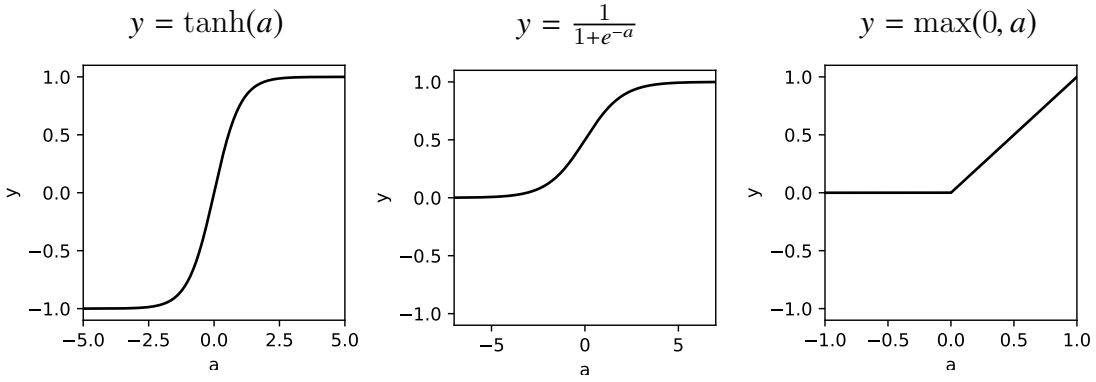


Figure 3.8 – Artificial Neuron

The trainable weight vector $\mathbf{w}$ determines the relevance of a neuron's input on the output. The trainable bias b determines the operating point in the non-linear activation function $\sigma(a)$.

The activation function $\sigma(a)$ models the biological counterpart in that the electrical spike traveling through the axon cannot be of infinite strength but will be saturated within a certain range. Common activation functions like the logistic sigmoid function or hyperbolic tangent activation keep the neuron's state y within a reasonable range (see figure 3.9 (a) and (b)).

However, in modern implementations of deep neural networks so called rectified linear units (ReLu) are often used instead [29,58,59] (see figure 3.9 (c)), as they can have a strong regularising effect, and are faster to train than hyperbolic tangent activation functions [27].



(a) Hyperbolic tangent (b) Logistic Sigmoid (c) Rectifier Linear Unit

Figure 3.9 – Different activation functions for artificial neurons. The hyperbolic tangent function (a) ranges from -1 to 1, whereas the logistic sigmoid (b) ranges from 0 to 1. The Rectifier Linear Unit (ReLu) clips negative values to 0.

3.2.2 Multi-Layer Perceptron (MLP)

Multiple interconnected artificial neurons form an artificial neural network. In feed-forward neural nets, such as so-called multi-layer perceptrons (MLP), neurons are organised in consecutive layers:

The first layer, also called input layer is followed by L so called hidden layers, with each hidden layer l consisting of $m^{(l)}$ parallel neurons.

The last layer $(L+1)$ is called the output layer, yielding the resulting output vector $\mathbf{y}^{(L+1)} = \left[y_1^{(L+1)}, y_2^{(L+1)}, \ldots, y_{m^{(L+1)}}^{(L+1)}\right]^T$ of a neural network.

Each layer's activations $a_i^{(l)}$ are transformed using non-linear activation functions $\sigma^{(l)}(a_i^{(l)})$ as described above (see figure 3.9) and fed to the inputs of the next layer $(l+1)$.

Or in other words, the states of the preceding layer $\mathbf{y}^{(l-1)} = \left[y_1^{(l-1)}, y_2^{(l-1)}, \ldots, y_{m^{(l-1)}}^{(l-1)}\right]^T$ are the inputs for the subsequent layer, as depicted in figure 3.10 l [60].

Each neuron's state y_i is given as

$$a_i^{(l)} = \sum_{j=1}^{m^{(l-1)}} w_{i,j}^{(l)} \cdot y_j^{(l-1)} + b_i^{(l)}, \qquad y_i^{(l)} = \sigma^{(l)}(a_i^{(l)}), \tag{3.26}$$

with bias $b_i^{(l)}$ and interconnection weight $w_{i,j}^{(l)}$.

In vector notation the activations for each layer can be rewritten as the product of a weight matrix $\mathbf{W}^{(l)}$ containing all interconnection weights $w_{i,j}^{(l)}$ between two consecutive layers l and $(l-1)$ and the outputs of the previous layer $\mathbf{y}^{(l-1)}$.

$$\mathbf{y}^{(l)} = \sigma^{(l)}\left((\mathbf{W}^{(l)})^T \mathbf{y}^{(l-1)} + \mathbf{b}^{(l)}\right). \tag{3.27}$$

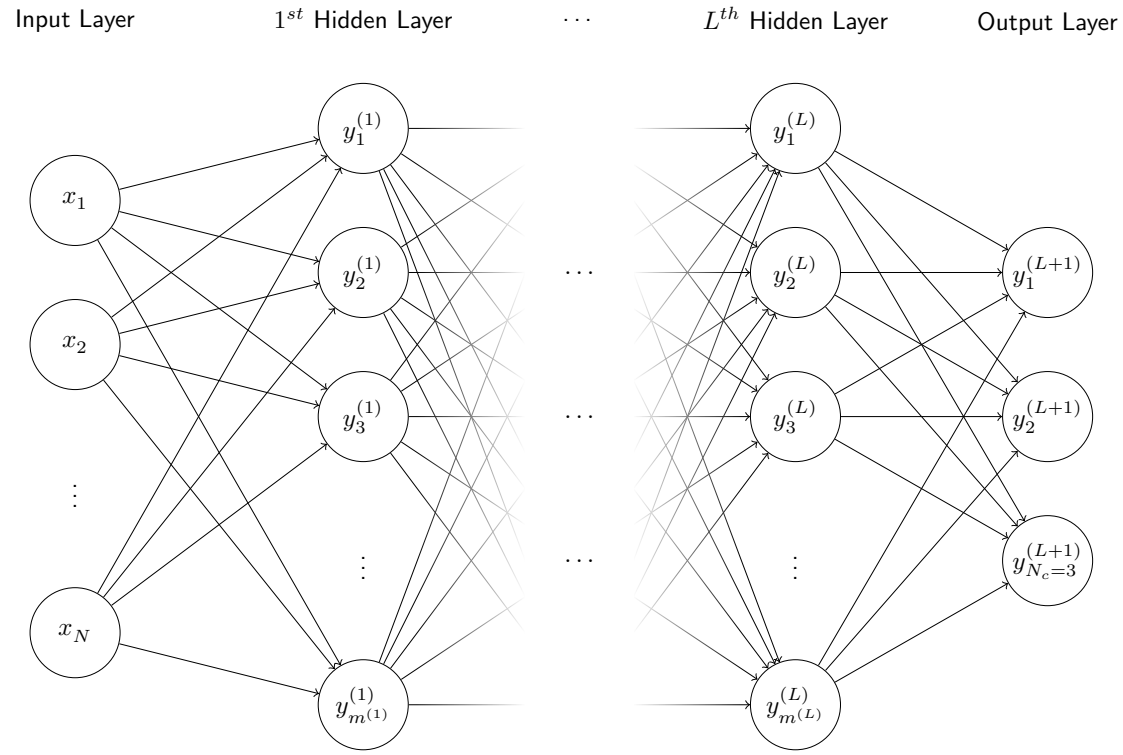


Figure 3.10 – Multilayer perceptron with N inputs, L hidden layers, each with $m^{(l)}$ neurons and $N_c = 3$ output neurons. Arrows represent weighted connections between neurons.

Figure 3.10 shows a multi-layer perceptron (MLP) with N inputs, L hidden layers and $N_c = 3$ output neurons.

Softmax Activation

In classification tasks, the output layer usually incorporates the so called softmax activation function, which normalises the sum of the output vector $\mathbf{y}^{(L+1)}$ to 1, so that each output y_c represents the a-posteriori detection probability of a certain class c [3.1]:

$$y_c = \frac{e^{a_c^{(L+1)}}}{\sum_{j=1}^{N_c} e^{a_j^{(L+1)}}}, \qquad \forall\ c = 1, \ldots, N_c \tag{3.28}$$

3.1. Sometimes it makes sense to use a logistic sigmoid activation instead, as this allows multiple classes to be detected at the same time [32].

3.2.3 Training a Neural Net

The successive adaptation of a neural network's parameters, such as interconnection weights $w_{i,j}^{(l)}$ and biases $b_j^{(l)}$ is called *training*. The training process aims to minimise a *loss function* $\mathcal{L}$, describing the deviation between the neural net's output $\tilde{\mathbf{y}} = \mathbf{y}^{(L+1)}$ and the desired target $\mathbf{y}$.

Loss function

The most common loss function for classification tasks using neural nets is the categorical cross entropy [61]. The categorical cross entropy is the Kullback-Leibler divergence of the estimated probability distribution $\mathbf{q}$ from the ground truth $\mathbf{p}$ and is given as

$$\mathcal{L}_{\mathbf{p}}(\mathbf{q}) = -\sum_{i=1}^{N_c} p_i \log(q_i), \tag{3.29}$$

where N_c is the number of categories or classes [62, 63].

The output of a neural net classifier is the estimated probability distribution $\tilde{\mathbf{y}}$, which aims to predict the target distribution $\mathbf{y}$. The target or ground truth $\mathbf{y}$ resembles a one-hot encoding of the class labels. So for the classes music, environmental noise and speech, the respective target distributions are given as

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

(a) Music (b) Env. Noise (c) Speech

A neural net's classification output $\tilde{\mathbf{y}}$ for a music segment might be the estimated a-posteriori probability distribution

$$\tilde{\mathbf{y}} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.95 \\ 0.02 \\ 0.03 \end{bmatrix}. \tag{3.30}$$

The categorical cross entropy of the predicted distribution $\tilde{\mathbf{y}}$ can then be computed as

$$\mathcal{L}_{\mathbf{y}}(\tilde{\mathbf{y}}) = -(1 \cdot \log(0.95)) \approx 0.051. \tag{3.31}$$

For a perfect prediction, the loss $\mathcal{L}$ will be 0. A classifier that guesses randomly will yield a categorical cross entropy of $\mathcal{L} = -(1 \cdot \log(1/3)) \approx 1.098$.

Parameter Optimisation
The goal of parameter optimisation is to minimise the loss $\mathcal{L}$ by iteratively adapting a neural net's trainable parameters, such as interconnection weights and biases. The trainable parameters are adapted layer by layer using the so-called back-propagation algorithm and an optimisation algorithm such as the stochastic gradient descent (SGD) or *Adam*.

Back Propagation Algorithm
When training a feed-forward neural net, the network's response to an input $\mathbf{x}$ is evaluated by propagating the initial information contained in $\mathbf{x}$ through the network layer by layer. This is called *forward propagation*. From the neural net's output $\tilde{\mathbf{y}}$ and the reference or target $\mathbf{y}$ a scalar loss $\mathcal{L}_{\mathbf{y}}(\tilde{\mathbf{y}})$ is derived.

To minimise the loss $\mathcal{L}$ by gradient descent or other optimisation algorithms, it is important to compute the partial derivatives of $\mathcal{L}$ for each trainable parameter in the network. To find the gradients $\nabla_{\mathbf{W}^{(l)}}\mathcal{L}$ and $\nabla_{\mathbf{b}^{(l)}}\mathcal{L}$ for each layer l, the error or loss $\mathcal{L}$ is *propagated back* through the network layer by layer starting with the output layer [64].

The underlying principle of the *back-propagation* algorithm is listed in pseudocode below (Algorithm 1). For a more detailed description of the back-propagation algorithm, please refer to [45, 64, 65].

Stochastic Gradient Descent
Different optimisation algorithms, also simply called optimisers, are used to calculate the weight and bias updates $\Delta\mathbf{W}$ and $\Delta\mathbf{b}$.

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} + \Delta\mathbf{W}^{(l)}, \qquad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} + \Delta\mathbf{b}^{(l)}. \tag{3.32}$$

For the sake of simplicity, all training parameters in $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ for each layer l are denoted as one parameter vector $\boldsymbol{\theta}$.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}. \tag{3.33}$$

The most common optimiser $o(\nabla_{\boldsymbol{\theta}}\mathcal{L})$ is the so-called stochastic gradient descent (SGD), which updates the weights into the direction of the steepest descent moving along the multi-dimensional error plane:

$$\Delta\boldsymbol{\theta} = o(\nabla_{\boldsymbol{\theta}}\mathcal{L}) = -\gamma \cdot \nabla_{\boldsymbol{\theta}}\mathcal{L} = -\gamma \cdot \frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}\,, \tag{3.34}$$

where $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is the gradient of the loss function $\mathcal{L}$ evaluated for a training example and γ is the learning rate or step size [45, p.150].

SGD is very sensitive to changes in the learning rate γ. If it is too small, finding the global minimum in the multi-dimensional loss plane will take very long. If it is two large, the algorithm might 'overshoot' the minimum in the error plane, resulting in an oscillation around it. To alleviate this problem, many different optimisation algorithms with adaptive learning rates, such as *Adam* have been developed.

Algorithm 1: Basic principles of back-propagation algorithm [45, 65].

1. Feed-forward:
feed the input $\mathbf{x}$ through the network layer by layer (*forward propagation*),
starting with the input layer $l = 1$
$\mathbf{y}^{(1)} \leftarrow \mathbf{x}$;
for $l = 2, \ldots, (L+1)$ **do**
  $\mathbf{y}^{(\mathbf{l})} = \sigma^{(l)}\left((\mathbf{W}^{(l)})^T \mathbf{y}^{(\mathbf{l-1})} + \mathbf{b}^{(l)}\right)$;
end

2. Compute loss:
compute the error or loss $\mathcal{L}_{\mathbf{y}}(\tilde{\mathbf{y}})$ from the neural net's output $\tilde{\mathbf{y}} = \mathbf{y}^{(L+1)}$
and the desired target $\mathbf{y}$.

3. Back-propagation:
propagate the error back through the network in order to obtain the partial derivatives or gradients $\mathbf{g}$ of the loss for each layer,
starting with the output layer $l = L + 1$
$\mathbf{g} \leftarrow \nabla_{\tilde{\mathbf{y}}} \mathcal{L}_{\mathbf{y}}(\tilde{\mathbf{y}})$;
for $l = L, L-1, L-2, ..., 1$ **do**
  $\mathbf{g} \leftarrow \left((\mathbf{W}^{(l+1)})^T \mathbf{g}\right) \odot \sigma'^{(l)}(\mathbf{a}^{(l)})$;

  compute rate of change of the loss with respect to the bias vector $\mathbf{b}^{(l)}$
  and weight matrix $\mathbf{W}^{(l)}$
  $\nabla_{\mathbf{b}^{(l)}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \mathbf{g}$;
  $\nabla_{\mathbf{W}^{(l)}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \mathbf{y}^{(l-1)} \mathbf{g}$;

  update trainable parameters by evaluating the optimiser function $o(\nabla \mathcal{L})$
  $\Delta \mathbf{b}^{(l)} = o(\nabla_{\mathbf{b}^{(l)}} \mathcal{L})$;
  $\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} + \Delta \mathbf{b}^{(l)}$;
  $\Delta \mathbf{W}^{(l)} = o(\nabla_{\mathbf{W}^{(l)}} \mathcal{L})$;
  $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} + \Delta \mathbf{W}^{(l)}$;
end

Adam
Adam was introduced by Kingma and Ba [66] in 2015. The name *Adam* is derived from the term adaptive moment estimation and combines advantages of other popular optimisation techniques, such as *AdaGrad* [67] and requires little to no tuning.

Adam keeps an exponentially decaying average of past gradients, estimating the first and second moment mean $\mathbf{m}$ and variance $\mathbf{v}$. This helps accelerating the gradient descent in the relevant direction, dampening oscillation around local

minima in the error plane, similar to momentum [3.2]:

$$\mathbf{m} \leftarrow \beta_1 \cdot \mathbf{m} + (1 - \beta_2) \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L} \tag{3.35}$$

$$\mathbf{v} \leftarrow \beta_1 \cdot \mathbf{v} + (1 - \beta_2) \cdot \nabla^2_{\boldsymbol{\theta}} \mathcal{L} \tag{3.36}$$

Before training, $\mathbf{m}$ and $\mathbf{v}$ are initialised to zero. To compensate the resulting bias in the early adaptation steps, they are corrected as follows:

$$\mathbf{m} \leftarrow \frac{\mathbf{m}}{1 - \beta_1} \tag{3.37}$$

$$\mathbf{v} \leftarrow \frac{\mathbf{v}}{1 - \beta_2} \tag{3.38}$$

The final update rule is then given as

$$\Delta\boldsymbol{\theta} = -\frac{\gamma}{\sqrt{\mathbf{v}} + \epsilon} \cdot \mathbf{m}, \tag{3.39}$$

with recommended parameters $\gamma = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ [66, 68].

Mini-Batch Training

When training on large amounts of data, calculating parameter updates by evaluating all training samples successively will take very long. In practice, a very good approximation of the gradient $\nabla\mathcal{L}$ can be computed evaluating a randomly sampled subset of training examples.

Optimisation algorithms evaluating a single sample at a time ($B = 1$) are called *stochastic* methods.

Optimisation algorithms using the entire training set for each parameter update are called *batch training* or deterministic gradient methods, with the batch size B being equal to the total number of observations O in the training set.

The so-called *Mini-batch* training algorithms fall somewhere in between, evaluating a small set of B samples at a time.

Multi-core setups with GPU-accelerated [3.3] computing allow for parallel evaluation of multiple samples, where the amount of memory and hard disk read speeds are usually the limiting factor. Small batch sizes have a regularising effect, reducing overfitting, but do not utilise multi-core setups well. Choosing the right batch size B comes down to finding a compromise between speeding up training and an accurate gradient estimation [45].

3.2. Momentum works by adding a fraction β of the previous update vector to the current update vector: $\Delta\boldsymbol{\theta} \leftarrow -\gamma \cdot \nabla_{\boldsymbol{\theta}}\mathcal{L} + \beta \cdot \Delta\boldsymbol{\theta}$

3.3. Graphics Processing Units (GPU) are specialised circuits for rendering image data. Incorporating many parallel processing units, they are very efficient for parallel analysis of large amounts of data.

Epochs
Usually, the training data will be presented to the neural net several times until the loss $\mathcal{L}$ has reached its minimum. The number of times, the full training data is presented to the neural net is called the number of epochs. Or put differently, one complete pass through all training examples is called an epoch. Usually training examples in the training set are shuffled before each epoch, so that successive training examples are not correlated. This prevents the network from getting stuck in local minima within the error plane [69].

Regularisation
The term regularisation describes techniques to guide the training process into the right direction and in consequence prevent overfitting.

One popular regularisation technique is the use of *dropout* between layers: Some randomly drawn interconnection weights are set to zero (typically a *dropout* propability of 25% - 50% is used), ensuring that the output at the end of the network does not depend on the state of only a few artificial neurons, which in consequence makes it less prone to overfitting [70].

Another very straight-forward regularisation method is *early stopping*. Instead of training a NN for a certain number of epochs, training will be stopped, as soon as the validation loss $\mathcal{L}_{\mathrm{y}}(\tilde{\mathbf{y}})$ stops improving [3.4]. This ensures that the NN does not overfit the training data. To do so, a test set is evaluated after each epoch to obtain the validation loss $\mathcal{L}_{\mathrm{y}}(\tilde{\mathbf{y}})$. Figure 3.11 shows the evolution of the training and evaluation loss for a NN trained for 35 epochs. As can be seen, validation loss stops improving after 22 epochs.

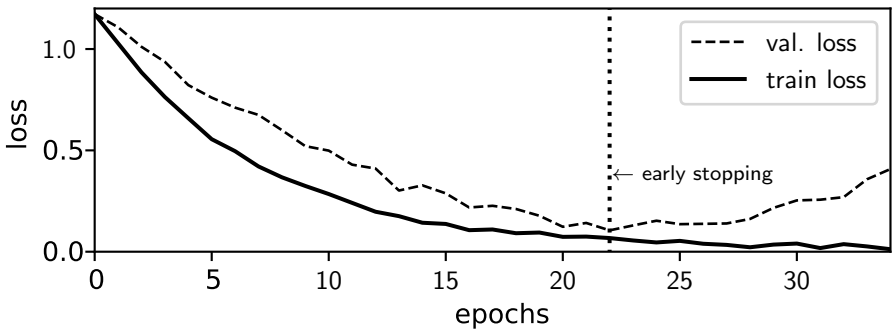


Figure 3.11 – Early stopping: Training is stopped when the validation loss stops improving.

Other regularisation efforts include unsupervised pre-training [71] and batch normalisation [72] but have not been further investigated for this thesis.

3.4. In practice, usually training is not stopped immediately but after a few epochs (also called patience), so that the overall trend of the validation loss is captured rather than small fluctuations.

3.2.4 Convolutional Neural Network (CNN)

Convolutional Neural Nets (CNN) are multi-stage architectures, consisting of several so-called convolutional layers and consecutive feature pooling / down-sampling layers followed by several fully-connected layers of artificial neurons. Originally developed and successfully used for image classification [27, 33], they have proven to work great for audio classification tasks [28, 29, 32].

Essentially, convolutional layers work by correlating segments of the input data with trained prototype filter kernels, yielding a high output score for matching patterns. By adding more convolutional layers, patterns of rising complexity can be extracted from the input image.

Figure 3.12 shows the model structure for a CNN with two convolutional layers with consecutive pooling layers and two fully-connected layers as a classifier.

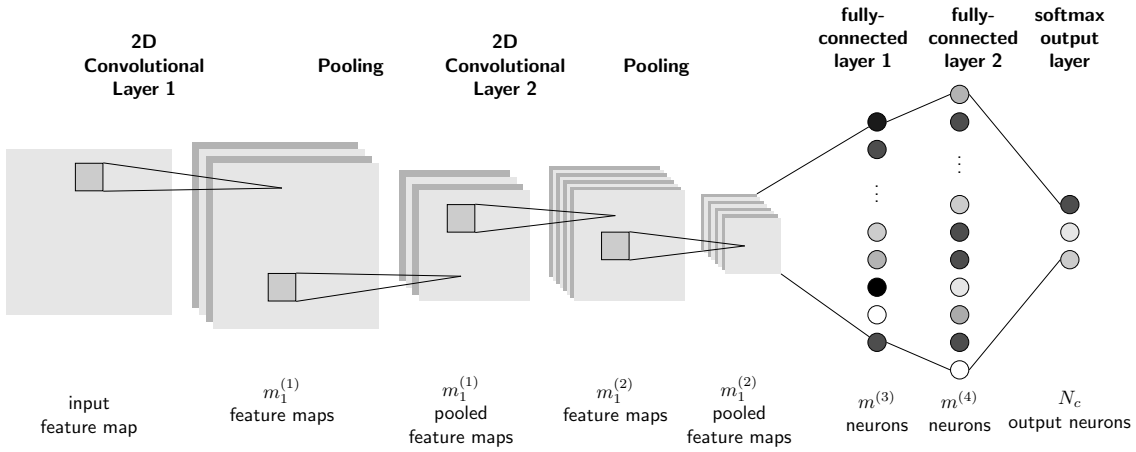


Figure 3.12 – Model structure of CNN featuring two convolutional layers with consecutive pooling and an MLP classifier with two fully-connected layers for image classification.

In the following, discrete convolution and its application in convolutional neural nets are explained.

Discrete Convolution

The convolutional operation for functions $x(t)$ and $k(t)$ is given as

$$s(t) = (x * k)(t) = \int x(\tau)k(t-\tau)d\tau. \tag{3.40}$$

Assuming that x and k are only defined at discrete intervals, the so called discrete convolution s_i can be defined as

$$s_i = (x * k)_i = \sum_{n=-\infty}^{\infty} x_i \cdot k_{i-n}. \tag{3.41}$$

Filter kernels can be one-, two- or three-dimensional, depending on the input data format. In our case two-dimensional kernels are used to extract relevant information from spectrograms.

With a two-dimensional kernel $\mathbf{K}$ of size $h_1 \times h_2$ and an input image $\mathbf{X}$ of size $m_2 \times m_3$ the convolution operation is defined as

$$s_{i,j} = (\mathbf{X} * \mathbf{K})_{i,j} = \sum_{m=1}^{h_1} \sum_{n=1}^{h_2} x_{i-m,j-n} \cdot k_{m,n}, \tag{3.42}$$

with the kernel $\mathbf{K}$ and input $\mathbf{X}$ being

$$\mathbf{K} = \begin{bmatrix} k_{0,0} & k_{0,1} & \cdots & k_{0,l_2} \\ k_{1,0} & k_{1,1} & \cdots & k_{1,l_2} \\ \vdots & \vdots & \ddots & \vdots \\ k_{l_1,0} & k_{l_1,1} & \cdots & k_{l_1,l_2} \end{bmatrix} \quad \text{and} \quad \mathbf{X} = \begin{bmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,m_2} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,m_2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m_1,0} & x_{m_1,1} & \cdots & x_{m_1,m_2} \end{bmatrix},$$

yielding the two-dimensional output $\mathbf{S}$ [45, p.322-324].

Figure 3.13 demonstrates the mechanics of discrete two-dimensional convolution: A small kernel (2×2) resembling a vertical edge is moved across the larger input image (5×5), and evaluated according to equation 3.42. As can be seen the vertical line is emphasised (as indicated by darker color) in the output image, while the horizontal line is getting washed out. Note that in this example no zero-padding was applied at the edges of the input image, resulting in reduced dimensions for the output image $\mathbf{S}$ (4×4).

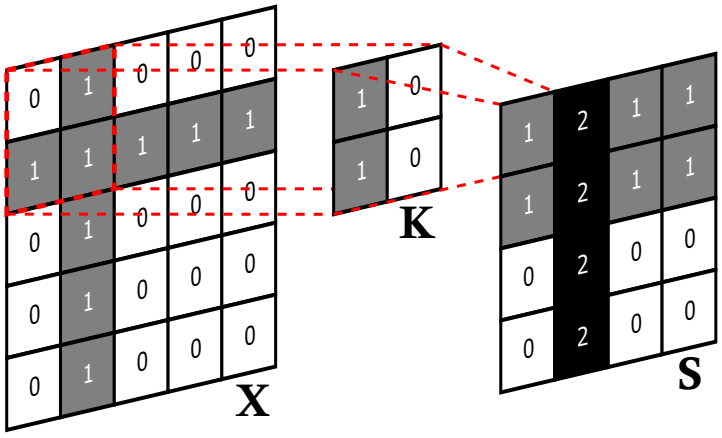


Figure 3.13 – Discrete two-dimensional convolution for input image $\mathbf{X}$ of size $n_2 \times n_3 = 5 \times 5$ with filter kernel $\mathbf{K}$ of size $h_1 \times h_2 = 2 \times 2$, resulting in an output image of size $m_2 \times m_3 = 4 \times 4$. The depicted filter kernel detects and emphasises vertical lines.

Convolutional Layer

The input of a two-dimensional convolutional layer l is a three-dimensional array with $n_1^{(l)}$ two-dimensional feature maps of size $n_2^{(l)} \times n_3^{(l)}$. For a spectrogram or black and white image, the number of channels or feature maps is $n_1^{(1)} = 1$ [3.5].

Each convolutional layer is composed of $m_1 \cdot n_1$ filter kernels $\mathbf{K}_{i,j}$, each of size $h_1^{(l)} \times h_2^{(l)}$, yielding $m_1^{(l)}$ two-dimensional feature maps of size $m_2^{(l)} \times m_3^{(l)}$ at the output. Each filter $\mathbf{K}_{i,j}$ detects a particular pattern at every location on the input $\mathbf{X}_j$ [33].

The output feature maps $\mathbf{Y}_i^{(l)}$ for layer l are given as the sum over the discrete convolutions of the trainable filter kernels $\mathbf{K}_{i,j}$ with the input feature maps $\mathbf{X}_j$ and a trainable bias $b_i^{(l)}$.

$$\mathbf{Y}_i^{(l)} = \sigma^{(l)} \left(b_i^{(l)} + \sum_{j=1}^{n_1^{(l)}} \mathbf{K}_{i,j}^{(l)} * \mathbf{X}_j^{(l)} \right), \tag{3.43}$$

with $i = 1, 2, \ldots, m_1$, $j = 1, 2, \ldots, n_1$ and non-linear transformation function $\sigma^{(l)}$.

Pooling Layer

So called pooling (also down-sampling) layers are used to reduce the resolution of a preceding convolutional layer's output feature maps $\mathbf{Y}_i^{(l-1)}$. Pooling works by moving an analysis window of size $p_1 \times p_2$ along the input image with a stride $s_1 \times s_2$ and evaluating the pooling function at each position, combining neighbouring pixels to a single value. The most popular pooling function is so called max pooling, where the analysis window is represented by its maximum value at the output, as demonstrated in figure 3.14

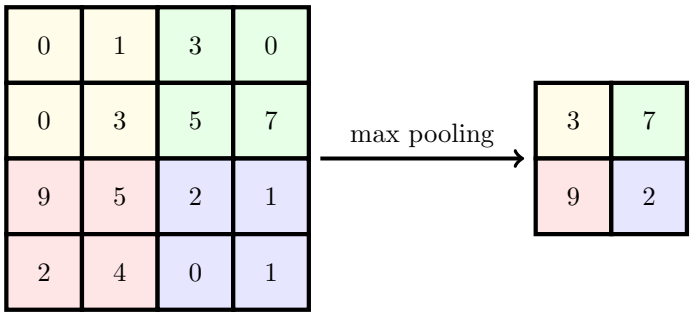


Figure 3.14 – *Max pooling* on a two-dimensional input image $\mathbf{X}$ of size $m_2 \times m_3 = 4 \times 4$ with pool size $p_1 \times p_2 = 2 \times 2$ and stride $s_1 \times_2 = p_1 \times p_2$ (no overlap).

3.5. For a colored RGB image (The RGB color space is an additive color model, describing a color in three channels, representing the colors red, green and blue.) the number of channels would be $n_1^{(1)} = 3$.

Fully-connected Layer
As mentioned before, the convolutional layers with consecutive pooling or down-sampling are followed by one or more so-called fully-connected layers, resembling a multi-layer perceptron (MLP) classifier. In other words, the outputs of the last convolutional layer resemble the features for classification with the MLP.

3.3 Evaluating a Classifier

To examine the performance of a classification algorithm, the trained model is applied to previously unseen data. Several measures can then be used to compare different algorithms, including the classification accuracy, precision and recall, as well as the so-called f-measure.

A way to visualise the performance of a classification algorithm on a test set, is the so called confusion matrix $\mathbf{M}$.

By definition, each element $m_{i,j}$ represents the number of samples known to be of class i and predicted as class j [73]. The diagonal ($i = j$) contains the number of correctly classified samples, the rest of the matrix contains the number of misclassifications ($i \neq j$).

$$\mathbf{M} = \begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,N_c} \\ m_{2,1} & m_{2,2} & \vdots & m_{1,N_c} \\ \vdots & \dots & \ddots & m_{2,N_c} \\ m_{N_c,1} & m_{N_c,2} & \dots & m_{N_c,N_c} \end{bmatrix} \tag{3.44}$$

Or in other words, each row i of $\mathbf{M}$ corresponds to the true label of a class. Each column j represents the predicted label respectively.

Figure 3.15 shows a confusion matrix for a speech, music and environmental noise classifier ($N_c = 3$). The evaluated test set contains N = 3000 samples, 1000 for each class. Out of 1000 music samples, 843 (84.3%) were predicted correctly, 103 (10.3%) were misclassified as environmental noise and 54 (5.4%) were mislabelled as speech. The second and third row represent the number of correct and false classifications for environmental noise and speech respectively.

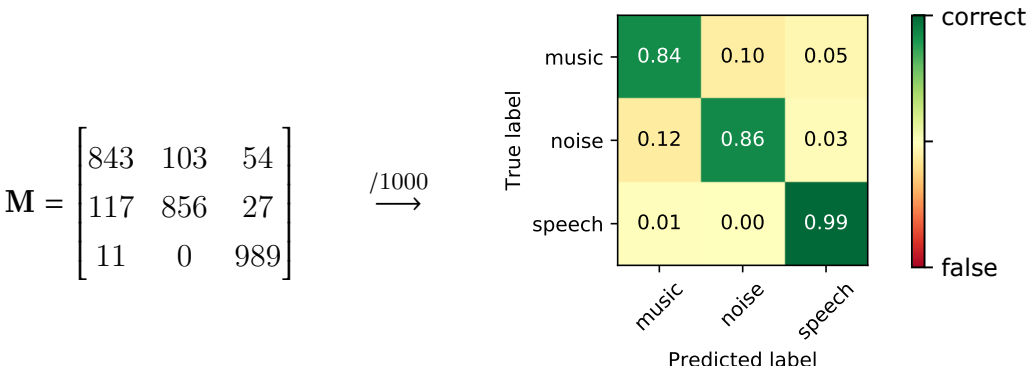


Figure 3.15 – The confusion matrix: Each row represents true class membership, each column represents predicted class membership respectively. The normalised confusion matrix (right) can be visualised with colours to make it easily interpretable. Green indicates correct classification, red indicates false classification.

From the the confusion matrix, several measures of classification performance can be derived.

Accuracy:
The classification accuracy a measures how many of the total N samples are classified correctly. Or put differently, the overall accuracy of a classifier is the number of samples in the main diagonal of the confusion matrix $\mathbf{M}$ divided by the total number of samples $N = \sum_{i=1}^{N_c} \sum_{j=1}^{N_c} m_{i,j}$.

$$a = \frac{\sum_{i=1}^{N_c} m_{i,i}}{N} \tag{3.45}$$

For the example in figure 3.15 the overall accuracy is $a = (843 + 856 + 989)/3000 \approx 90\%$.

Precision
Precision p_c (also called positive predictive rate) is defined as the ratio of the number correct predictions and the total number of predictions of class c (sum over column).

$$p_c = \frac{m_{c,c}}{\sum_{i=1}^{N_c} m_{i,c}} \tag{3.46}$$

For our example, the precision for music is $p_1 = 843/(843 + 117 + 11) \approx 86.28\%$.

The overall precision for the evaluated classifier is the average subject to the number of evaluated samples for each class c (sum over row).

$$p = \sum_{c=1}^{N_c} p_c \frac{\sum_{j=1}^{N_c} m_{c,j}}{N} \tag{3.47}$$

For our example, the overall precision is $p = (86.28\% + 89.35\% + 91.91\%)/3 \approx 90\%$.

Recall

Recall r_c (also called hit rate) is the ratio between the number correct predictions of class c and the number of samples for class c (sum over row).

$$r_c = \frac{m_{c,c}}{\sum_{j=1}^{N_c} m_{c,j}} \tag{3.48}$$

For our example, the recall for music is $r_1 = 843/(843 + 103 + 54) = 84\%$.

Analogous to eq. 3.47, the overall recall for the evaluated classifier is given as

$$r = \sum_{c=1}^{N_c} r_c \frac{\sum_{j=1}^{N_c} m_{c,j}}{N}. \tag{3.49}$$

For our example, the overall recall is $r = (84.3\% + 85.6\% + 98.9\%)/3 \approx 90\%$. As can bee observed, when the number of training examples is equal for each class, the overall recall r is equivalent to the accuracy.

F-score

The so-called F-score is the harmonic mean of precision and recall. It is 1 for perfect classification, i.e. a precision and recall of 1, and 0 in worst case.

$$F_c = 2 \cdot \frac{p_c \cdot r_c}{p_c + r_c} \tag{3.50}$$

For our example, the F-score for music F_1 is $2 \cdot 87 \cdot 84\%/(87\% + 84\%) \approx 86\%$.

As with precision and recall, the overall F-score is computed with regard to the distribution of samples among classes.

$$F = \sum_{c=1}^{N_c} F_c \frac{\sum_{j=1}^{N_c} m_{c,j}}{N}. \tag{3.51}$$

For our example, $F = (86\% + 87\% + 96\%)/3 \approx 89\%$.

4 | MUSPEN Dataset

As described in chapter 3, machine learning (ML) algorithms adapt their parameters based on experience gained by evaluating so-called training data. The performance of a ML algorithm is validated using separate test data.

The Music, Speech and Environmental Noise (MUSPEN) dataset consists of 150h hours of *'clean'* speech, music and noise data for training. To overcome the lack of training data recorded in real acoustic environments, the dataset was augmented using a room simulation procedure (see section 4.5), yielding another 150h of *'degraded'* training data.

In order to evaluate a classifier's performance, the MUSPEN dataset contains 30h of *'clean'* test data. Additionally, 1h of test data was recorded with a far-field microphone array and manually labelled, resembling real world data (see section 4.4) . This test dataset is referred to as the *'target'* test data.

Table 4.1 lists the amount of training and test data for each class.

	TRAIN		TEST	
class	*'clean'*	*'degraded'*	*'clean'*	*'target'*
Music	50h	50h	10h	20m
Speech	50h	50h	10h	20m
Env. Noise	50h	50h	10h	20m
total	150h	150h	30h	1h

Table 4.1 – The Music, Speech and Environmental Noise (MUSPEN) dataset. This table shows the duration of training and test data in hours.

Training data includes music, speech and environmental noise signals from various sources, ensuring the training data features a wide variety of examples.

As the different sources provide audio data in different (compressed and uncompressed) data formats, all audio data was re-sampled to $f_s = 16$kHz and 16bits and stored in .wav-files for further processing. For multi-channel audio files, only the first channel was kept.

Section 4.1 describes the *'clean'* music data collected for training and evaluation. Sections 4.2 and 4.3 present speech and environmental noise data respectively.

4.1 Music Data

The MUSPEN dataset includes a total of 60h of *'clean'* music data of which 50h are used for training and 10h are used for evaluation purposes. As described below, training and test data come from different sources, ensuring that audio from the test set is not included in the training dataset and vice-versa.

Training Data
Training data is taken from the GC16UX song dataset provided by the International Music Information Retrieval Systems Evaluation Laboratory (IMIRSEL) as part of the MIREX challenge 2016 [74]. The dataset contains 10,000 songs in MP3 format from Jamendo.com [4.1], a web-service hosting royalty-free music.

The dataset includes meta-data, such as each track's genre, artist and duration. Figure 4.1 shows the distribution of music data by genre [4.2]. As most music in the original dataset is labeled as electronic music, a subset of 50h was drawn from the full dataset, that more evenly represents different styles of music (6.25h per genre category).

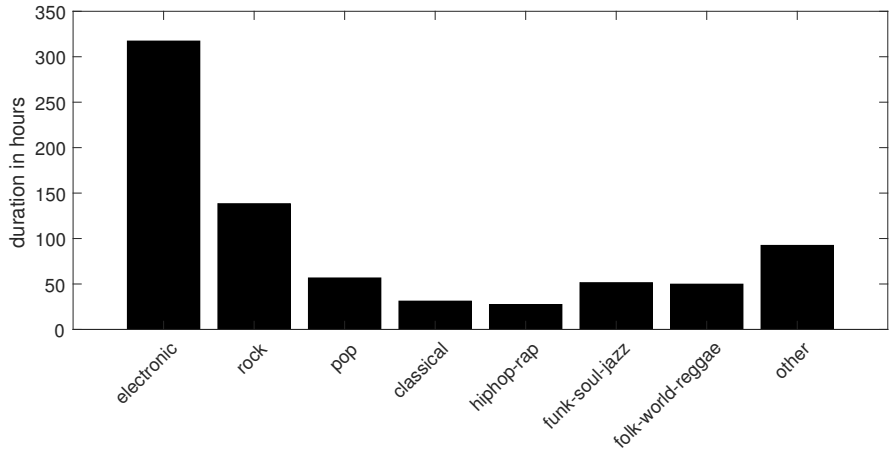


Figure 4.1 – Distribution of combined genres in the GC16UX song dataset [74]. Most music in the database is labeled as electronic music (317.2h). Some genres, such as funk, soul and jazz were combined into one category.

Test Data
Test data was gathered by downloading popular music mixes from YouTube.com in MP3 format with a total duration of 10h. As with the training data, it includes music from various genres. More specifically the dataset contains approximately 1h of audio data for each of the following categories: hip-hop, classic rock, heavy metal, jazz, classical music, funk, raggae, electronic dance music, pop and folk.

4.1. *https://www.jamendo.com/* - accessed in June 2017
4.2. Genres annotations provided with the dataset were parsed and combined, so that each genre consists of at least 450 audio files.

4.2 Speech Data

Speech data was taken from the LibriSpeech database, curated by Panayotov et al. [75] for training and evaluating speech recognition systems. The dataset consists of approximately 1,000h of English speech from hundreds of different speakers, as uncompressed .flac-files sampled at 16kHz. The audio data is collected from audio books provided by libriVox.org [4.3], an online-service hosting free public domain audio books read by amateurs.

Training Data
50h of speech recordings were taken for training, including 60 female and 60 male speakers into the training set, around 25 minutes of speech recordings for each speaker. All speech signals feature one single speaker. All spoken text is in the English language.

Due to the lack of available data, no children are included in the training data set. As already mentioned in chapter 2.1, special phonations such as screaming, crying or whispering are not included in the training data set.

Test Data
10h of speech data from various sources was collected, including 40 speakers from the librivox database, that are not included in the training set (around 8 minutes per speaker), as well as manually curated and segmented politician speeches and TED science talks [4.4]. The test data contains approximately 50% female and 50% male speech. Around 25 minutes is children speech.

4.3. *https://librivox.org* - accessed in June 2017.
4.4. TED (Technology, Entertainment, Design) is an annual conference in California, USA. Speeches from TED conferences are available online, featuring a wide variety of topics and speakers. *https://www.ted.com* - accesed in June 2017

4.3 Environmental Noise Data

To our knowledge, no large-scale environmental noise databases were publicly available at the time of collecting audio data. Existing popular noise datasets, such as NOISEX-92 or AURORA-2 typically only feature very little data [76].

Train Data
To cover a wide variety of sounds from different acoustic environments, several publicly available datasets have been combined:

The dataset provided by Stowell et al. for the *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events* (DCASE) [77] features audio data from different acoustic environments, including different outdoor and indoor locations [4.5].

The *QUT-NOISE* database was developed by Dean et al. in 2010 [76] to evaluate voice activity detection algorithms. It features audio data recorded in different domestic and public environments [4.6].

Stork et al. have recorded kitchen noise for audio-based human activity recognition with robots [78]. The audio dataset *Freiburg 106* is publicly available [4.7].

The *TUT Acoustic Scenes 2017* [79] dataset features audio segments from 15 different indoor and outdoor locations [4.8] with a total 52 minutes of audio.

The *Diverse Environments Multichannel Acoustic Noise Database (DEMAND)* [80] provides 1.5h of multi-channel recordings in different outdoor and indoor environments [4.9]. The dataset is publicly available [4.10].

Additionally, audio data from the Auditory Perception Lab at Carnegie Mellon University [81], Beltran et al. at CICESE [82] [4.11] and the CNBC Stimulus Repository [83] was included into the training dataset.

Furthermore, manually curated audio data downloaded from Freesound.org [4.12] and Soundcloud.com [4.13] was added to obtain a total of 50h of training data.

Test Data
10h of environmental noise was collected from various sources (Freesound.org,

4.5. acoustic scenes included are: bus, busy street, office, open air market, park, quiet street, restaurant, supermarket, tube, tube station
4.6. acoustic scenes included are: café, home, street, car and reverb.
4.7. *http://www.csc.kth.se/~jastork/pages/datasets.html* - accessed in July 2017
4.8. acoustic scenes included are: bus, cafe / restaurant, car, city center, forest path, grocery store, home, lakeside, library, metro station, office, residential area, train, tram, urban park
4.9. acoustic scenes included are: domestic, nature, office, public, street, transportation
4.10. *http://parole.loria.fr/DEMAND/* - accessed in July 2017
4.11. available at *http://sound.natix.org* - accessed in July 2017
4.12. *https://freesound.org* - accessed in July 2017
4.13. *https://soundcloud.com* - accessed in July 2017

Soudcloud.com, Youtube.com ...), ensuring that no data in the training set is included in the test set and vice versa. To make sure, noise data resembles environments that are likely to occur in real-world scenarios, sounds of different categories were collected. The dataset contains approximately 1h of audio data for each of the following environments/categories: bathroom, in-car traffic noise, city, kitchen, miscellaneous living, office, outdoor, party (no music or isolated speech) [4.14], weather and workshop.

Due to the diverse nature of the environmental noise sounds, the data is rather inhomogeneous, especially when compared to the speech and music data.

4.4 Mic-Array Recordings - *'Target'* Data

To investigate the classifiers' performance in real-world scenarios, a small so-called *'target'* test set was recorded with the microphone array of the *Harman Kardon Invoke* voice-activated loudspeaker (see figure 1.1).

The audio data was recorded in different domestic and business environments, including a kitchen, a living / bed room, a small bathroom and an office environment.

Speech data includes six different speakers, including the author. The data features three female and three male speakers, all speaking in German. Music data includes music of various genres played back by stereo systems of different sizes and at different levels. Environmental Noise data includes different everyday activities like hoovering, doing the dishes and taking a shower.

In total 1h of *'target'* test data was used for validation purposes, 20 minutes for each class.

4.5 Data Augmentation - 'Degraded' Data

The goal of this thesis is to develop an algorithm that effectively classifies an audio data stream recorded with a far-field microphone array in real-life scenarios. Unfortunately, no labelled audio data recorded with the far-field microphone array was readily available.

As shown by Salamon et al. [28], the lack of training data can be overcome using data augmentation techniques. In our case, the clean audio data in the training set was processed, simulating the playback and recording of the *'clean'* training data in different acoustic environments. This way, training data is augmented, without actually having to record and manually label large amounts of data.

More specifically, the training audio data was chopped into segments of 10s each.

4.14. This category includes so-called babble noise, which occurs when many individuals talk simultaneously.

Each segment was then convolved with one of 15 room impulse responses from different indoor locations [4.15] and one of seven microphone impulse responses (as measured for the *'target'* microphone array in an anechoic chamber). The basic signal flow for signal degradation is shown in figure 4.2.

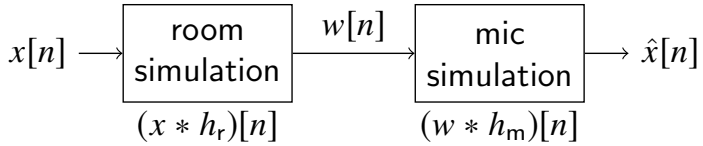


Figure 4.2 – Degrading audio input signal x by convolving it with the impulse response of a room h_r and the impulse response of a microphone h_m.

Each audio segment of 10s was then normalised after degradation. The resulting audio data yields the *'degraded'* train data (see table 4.1).

4.15. Room impulse responses come from small and medium sized indoor locations with reverberation times RT_{60} between 0.4s and 1.5s. Room impulse responses can be found at *http://www.openairlib.net/auralizationdb* and *https://github.com/idiap/acoustic-simulator/tree/master/impulse-responses-original/spaces* - accessed August 2017

5 | Machine Learning Approach

As described in chapter 1.2, feature-based machine learning algorithms have been used successfully for audio classification [9, 11, 17], using different statistical classification procedures from the field of artificial intelligence (AI).

While many classification algorithms, such as Gaussian mixture models (GMM) and the k-nearest neighbours (kNN) algorithm, yielded promising results, support vector machines (SVMs) have proven to perform better for multi-class audio classification tasks [84].

For this thesis, a simple Gaussian naive bayes (NB) algorithm is evaluated, forming the baseline to compare other classifiers against. Additionally a kNN classifier and a non-linear SVM with radial basis function (RBF) kernel are investigated.

The underlying principles are presented in chapter 3. This chapter describes the descriptive audio-features and the implementation of the classifiers evaluated for this thesis.

5.1 Descriptive Audio Features

The characteristics of speech and music signals described in chapter 2, can be exploited to define so-called descriptive features, which are more compact representations of the respective audio signal.

In accordance with Khan et al. [10], audio features are extracted at frame-level and clip-level. An audio frame is typically some tens of milliseconds long. Shorter frames allow for a more accurate tracking of transient events. Longer frames on the other hand will increase resolution for spectral analysis.

Classification can be carried out on frame-level, leading to adequate results. However, quite many frames will not be characteristic of the particular class and likely lead to misclassification [85]. It thus makes sense to integrate frame-level features over time into what is referred to as clip-level features. A clip is typically one to several tens of seconds long and characterises the distribution of frame-level features over time, as will be described in chapter 5.1.2. Audio data is segmented so that the content within a clip belongs to one class [10].

Section 5.1.1 describes investigated frame-level features. Section 5.1.2 describes temporal integration techniques used to obtain clip level features. The final features set used for classification is then obtained by ranking different clip-level features, as described in section 5.1.3.

5.1.1 Frame-level Features

A total number of 26 frame-level features, as listed in table 5.1, are evaluated for this thesis, including features derived from the time signal $x[n]$, a mid-level spectral representation $X(k)$, as well as features in the cepstral domain.

These features were chosen, as they are well-researched and many of them have been proven effective for various audio classification tasks [9, 11, 17]. Some are readily available in the *librosa* audio analysis toolbox [86] and the Speech *Signal Processing Toolkit* (SPTK) [87]. Others have been implemented based on the *Timbre Toolbox* [88].

Symbol	ID	Feature	Page	Source	#
ZCR	zcr	Zero-Crossing-Rate	61	librosa [86]	1
STE	ste	Short-Time Energy	61	librosa [86]	1
μ_1	spec_centroid	Spectral Centroid	64	librosa [86]	1
μ_2	spec_spread	Spectral Spread	64	librosa [86]	1
μ_3	spec_skew	Spectral Skewness	65	Timbre [88]	1
μ_4	spec_kurt	Spectral Kurtosis	65	Timbre [88]	1
f_{ro}	spec_rolloff	Spectral Roll-Off	65	librosa [86]	1
SF	spec_flux	Spectral Flux	65	Timbre [88]	1
H_C	chroma_ent	Chromatic entropy	67		1
$\mathbf{c}$	mfccs	Mel Frequency Cepstral Coefficients	70	librosa [86]	12
f_0	f0	Fundamental Frequency	71	SPTK [87, 89]	1
Harm	harmonicity	Harmonicity	71	Timbre [88]	1
T_1, T_2, T_3	tristimulus	Tri-stimulus	72	Timbre [88]	3
Total					26

Table 5.1 – Frame-level features. Source is listed if feature was readily available.

All training and test data (see chapter 4) is sampled at f_s = 16 kHz. Frame-level features are evaluated for a frame length of 1024 samples (64ms) and a hop size of 160 samples (10ms), yielding 100 feature values per second. Spectral features were computed, using a Fast Fourier Transform (FFT) with a resolution of 1024 frequency bins (see chapter 5.1.1.2).

5.1.1.1 Time-based features

Zero-Crossing Rate (ZCR)
As the name suggests, the zero-crossing rate (ZCR) measures how often a signal crosses zero within a certain amount of time. It is high for noisy sounds and lower for periodic signals, such as voiced speech segments or musical tones [90].

The ZCR for a frame-size N is given as

$$\mathrm{ZCR}[t] = \sum_{n=t-N}^{t} (1 - \delta(\mathrm{sign}(x[n-1]), \mathrm{sign}(x[n]))), \tag{5.1}$$

with the Kronecker delta $\delta(a, b)$ and the sign function $\mathrm{sign}(x)$ given as

$$\delta(a,b) = \begin{cases} 1 & \text{for } a = b, \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \mathrm{sign}(x) = \begin{cases} 1 & \text{for } x \geq 0, \\ 0 & \text{otherwise .} \end{cases} \tag{5.2}$$

Short-Time Energy (STE)
The short-time energy (STE) estimates the signal energy at a given time. Contrary to the ZCR, the STE is typically higher for voiced sounds and lower for unvoiced sounds [90].

It is computed by evaluating the root mean square (RMS) energy of each frame of length N:

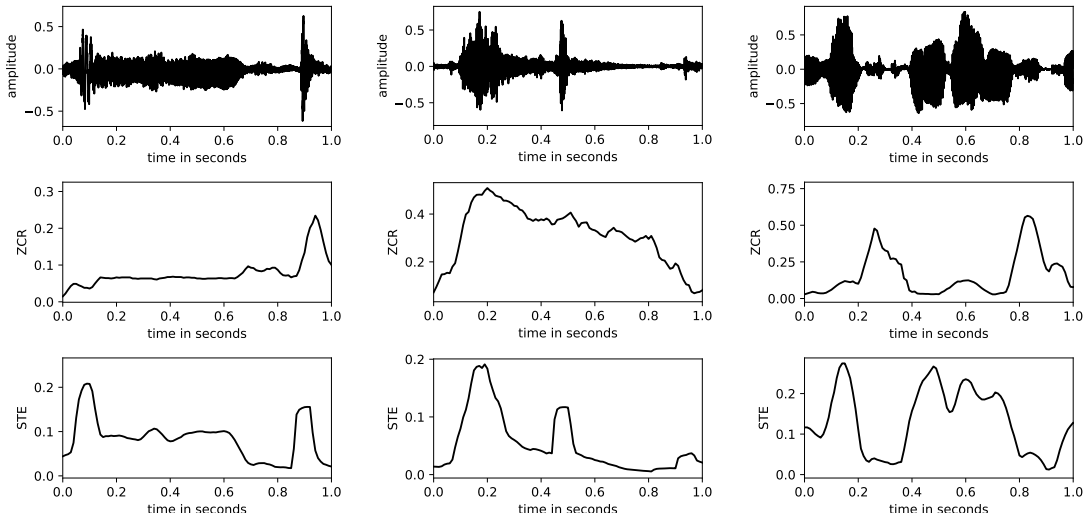
$$\mathrm{STE}[t] = \sqrt{\frac{1}{N} \sum_{n=t-N}^{t} x^2[n]}. \tag{5.3}$$

Figure 5.1 shows the logarithmic magnitude spectrogram $|X[k,t]|_{\mathrm{dB}}$ (see chapter 5.1.1.2 below), the ZCR, and STE for a music, environmental noise and speech signal respectively.

As expected for a pitched music signal (a), the ZCR is generally low and stationary for longer segments. For the speech signal (c), the ZCR fluctuates more rapidly, with higher values for unvoiced speech segments and lower for voiced segments.

For the speech signal (c) the STE fluctuates with each syllable and is higher for voiced and lower for unvoiced, noise-like speech segments. As speech contains pauses, the average STE is lower than for music (a).

Due to the heterogeneous nature of environmental noise sounds, the shown audio features for environmental noise (b) are less characteristic and conclusive.



(a) music (b) env. noise (c) speech

Figure 5.1 – audio waveform, zero-crossing rate (ZCR) and short-time energy (STE) for 1s of (a) music, (b) environmental noise and (c) speech.

5.1.1.2 Spectral Features

Spectral analysis describes a signal's energy distribution along frequency. The spectrum of a time signal x(t) is given as the *Fourier Transform* $\mathcal{F}\{x(t)\}$:

$$X(\omega) = \mathcal{F}\{x(t)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt, \tag{5.4}$$

with frequency ω. For discrete signal $x[n]$ of finite length N, we can compute the *Discrete Fourier Transform* (DFT) for each spectral component k:

$$X[k] = \mathrm{DFT}\{x[n]\} = \sum_{n=1}^{N} x[n]e^{-j\frac{2\pi}{N}nk} = |X[k]| \cdot e^{j\varphi[k]} \tag{5.5}$$

The DFT is complex-valued with magnitude $|X[k]|$ and phase $\varphi[k]$, given as

$$|X[k]| = \sqrt{\Re\mathrm{e}\{X[k]\}^2 + \Im\mathrm{m}\{X[k]\}^2}, \qquad \varphi[k] = \arctan\left(\frac{\Im\mathrm{m}\{X[k]\}}{\Re\mathrm{e}\{X[k]\}}\right), \tag{5.6}$$

where $\Re\mathrm{e}\{X[k]\}$ denotes the real part and $\Im\mathrm{m}\{X[k]\}$ the imaginary part of $X[k] = \Re\mathrm{e}\{X[k]\} + j \cdot \Im\mathrm{m}\{X[k]\}$ [91].

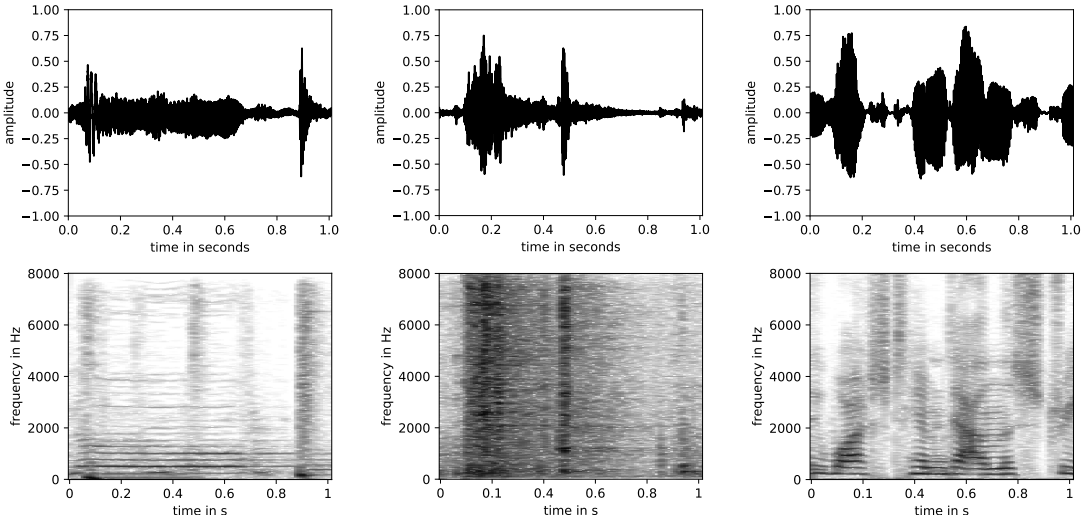
The *power spectrum* $S[k] = |X[k]|^2$ (also *spectral density*) describes the signal energy as a function of frequency, estimating how much each spectral component k contributes to the signal. Often, the spectrum is evaluated in the logarithmic domain, yielding the *logarithmic power spectrum* (LPS) $S_{\mathrm{dB}}[k]$ in decibels:

$$S_{\mathrm{dB}}[k] = 20 \cdot \log_{10}|X[k]| \tag{5.7}$$

The so-called *Short-Time Fourier Transform* (STFT) is the DFT computed for each time frame t. The signal is evaluated segment-wise. Each segment or frame is multiplied with a window function $w[n]$ (i.e. a Hann window). The STFT

$$X[k,t] = \sum_{n=0}^{N} x[t+n] \cdot w[n] e^{-j\frac{2\pi}{N}kn}, \tag{5.8}$$

yields the *Spectrogram* $X[k,t]$ evaluated for each time instance t and frequency $k = 1, 2, \ldots, N$. In practice, the *Fast Fourier Transform* (FFT) is used, which is the fastest technical implementation of the STFT for power-of-2 frame lengths. Figure 5.2 shows the time-signal $x[n]$ (top) and the logarithmic magnitude spectrogram $|X[k,t]|_{\text{dB}}$ (bottom) for a music signal (a), environmental noise (b) and a speech signal (c).



(a) music (b) env. noise (c) speech

Figure 5.2 – audio waveform and FFT spectrogram $|X[k,t]|_{\text{dB}}$ for 1s of (a) music, (b) environmental noise and (c) speech.

The music signal's spectrogram (a) features characteristic stationary horizontal lines, which represent the musical notes and their harmonics, as well as some vertical lines, corresponding to broad-band transient noise-like events like drum hits.

The speech signal's spectrogram (c) is more sparse and features a characteristic succession of segments with horizontal lines (voiced speech segments), pauses and broad-band transient segments.

In the following, features derived from the magnitude spectrum will be presented, describing the spectral shape with only a few values.

The harmonic structure of pitched signals, such as tonal music, voiced speech segments or pitched environmental noise sounds (i.e. horizontal lines in the

spectrogram) can be exploited for additional features, as will be described in section 5.1.1.4.

Spectral Centroid

The first statistical moment of the spectrum is the spectral centroid μ_1.

For pitched sounds, the spectral centroid is a good indicator for a sound's 'richness in harmonics' and its evolution over time. A sound with strong harmonics has a higher center of gravity compared to a sound with less prominent harmonics. The center of gravity is linked to the fundamental frequency f_0 of a sound. [91, p.363].

For unpitched sounds, the spectral centroid is an indicator for the 'brightness' of a sound. Brighter noise sounds (e.g. 'hissing' sounds or some affricatives in speech [$\mathrm{s}, \mathrm{f}, \int, ...$]) have a higher center of gravity than more dampened sounds.

The spectral centroid is defined as the geometric centroid (also called center of gravity) of the power spectrum $S[k]$ (or magnitude spectrum $|X[k]|$). First, a normalised spectral density is defined as p_k:

$$p_k[t] = \frac{S[k,t]}{\sum_{k=1}^{K} S[k,t]} \tag{5.9}$$

The spectral centroid $\mu_1[t]$ evaluated at each time step t is then given as

$$\mu_1[t] = \sum_{k=1}^{K} f_k \cdot p_k[t], \tag{5.10}$$

where f_k is the frequency of the spectral component k in Hertz.

Spectral Spread

The second statistical moment of the spectrum is called spectral spread. Wide-band (noisy) signals will have a greater spectral spread compared to more narrow-band sounds.

It is defined as the variance of a spectral distribution μ_2 around the spectral centroid μ_1.

$$\mu_2[t] = \sqrt{\sum_{k=1}^{K} (f_k - \mu_1[t])^2 \cdot p_k[t]} \tag{5.11}$$

Spectral Skewness

The third statistical moment of the spectrum is called spectral skewness μ_3. It describes the shape, or more precisely the asymmetry of the spectral distribution around the spectral centroid $\mu_1[t]$.

Negative spectral skewness $(\mu_3 < 0)$ indicates, that more energy is located below the spectral centroid μ_1, positive skewness $(\mu_3 > 0)$ indicates, more energy is

located in higher frequencies, above μ_1 [88].

$$\mu_3[t] = \frac{1}{\mu_2^3[t]} \left(\sum_{k=1}^{K} (f_k - \mu_1[t])^3 \cdot p_k[t] \right) \tag{5.12}$$

Spectral Kurtosis

Spectral kurtosis μ_4, the fourth statistical moment, is a measure of the peakedness or flatness of the spectrum. A lower number indicates a flatter, i.e. more evenly distributed spectrum.

Broad-band noise sounds will have very low values for μ_4, pitched sounds with weak harmonics on the other hand will have large values for μ_4.

$$\mu_4[t] = \frac{1}{\mu_2^4[t]} \left(\sum_{k=1}^{K} (f_k - \mu_1[t])^4 \cdot p_k[t] \right) \tag{5.13}$$

Spectral Roll-Off

The spectral roll-off frequency f_{ro} is the frequency below which 85% of the signal energy is located [11,86]. This measure is related to the skewness μ_3 of the spectral shape (see above) in that it is higher for left-skewed and lower for right-skewed spectra.

Calculating the spectral roll-off frequency comes down to finding the smallest spectral component $\hat{k} \in \{1, 2, \ldots, K\}$ for which equation 5.14 holds true. The roll-off frequency f_{ro} is then given as the frequency of spectral component $\hat{k}$ in Hertz.

$$\sum_{i=1}^{\hat{k}} p_i[t] \leq 0.85 \cdot \sum_{j=1}^{K} p_j[t]. \tag{5.14}$$

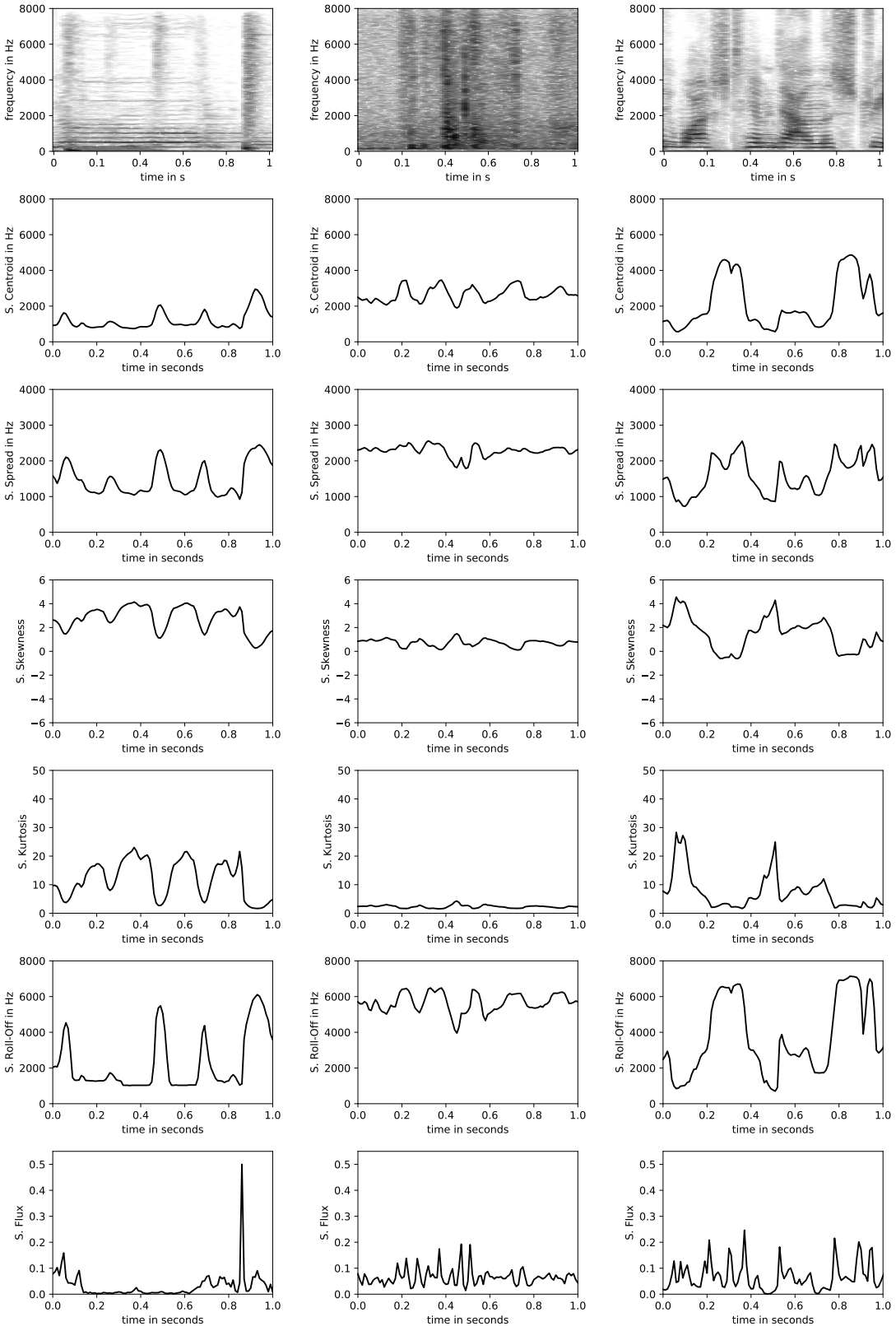
Spectral Flux

The so-called spectral flux SF is a measure of novelty and is known for it's ability to detect onsets in music and speech signals [92].

SF estimates the amount of spectral variation over time by correlating two successive frames according to [90].

$$\text{SF}[t] = 1 - \frac{\sum_k^K p_k[t-1] \cdot p_k[t]}{\sqrt{\sum_k^K p_k^2[t-1]} \cdot \sqrt{\sum_k^K p_k^2[t]}} \tag{5.15}$$

Figure 5.3 shows the spectral moments μ_1, μ_2, μ_3 and μ_4, as well as the spectral roll-off f_{ro} and flux spectral SF for music, environmental noise and speech signals respectively.



(a) music (b) env. noise (c) speech

Figure 5.3 – Spectrogram $|X[k,t]|_{\text{dB}}$ and its statistical moments spectral centroid μ_1, spread μ_2, skewness μ_3 and kurtosis μ_4, as well as the features spectral roll-off f_{ro} and spectral flux SF for 1s of (a) music, (b) environmental noise and (c) speech.

Chromagram

The perception of musical pitch can be modeled as a helix, showing the relation of perceived musical pitch or chroma and frequency. Moving upwards, one full rotation of the helix corresponds to one octave, i.e. doubling in frequency. Pitched sounds being exactly one (or multiple) octaves apart, are perceived as the same musical tone (see figure 5.4) [93].

Features exploiting this behaviour, so-called chroma features are well-established for analysing and comparing music signals [94].

Essentially, the chromagram combines the signal energy of the entire spectrum into 12 bins, representing the distinct semitones of the musical octave: $C[k,t]$ with $k = 1, 2, \ldots, 12$ [95].

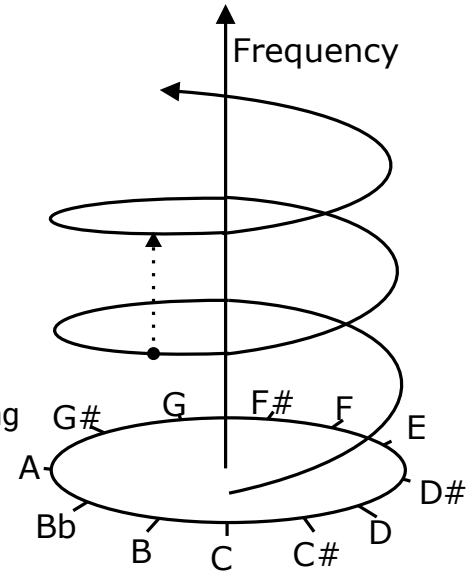


Figure 5.4 – Perception of musical pitch [93].

Chromatic Entropy

As the absolute values of pitch and harmony of a sound do not help to distinguish its type (speech, music or environmental noise), the chromatic entropy H_C is proposed.

For music signals, H_C is expected to be higher than for speech signals, so the chromatic entropy should be a good feature for speech music discrimination.

The chromatic entropy H_C is computed as follows.

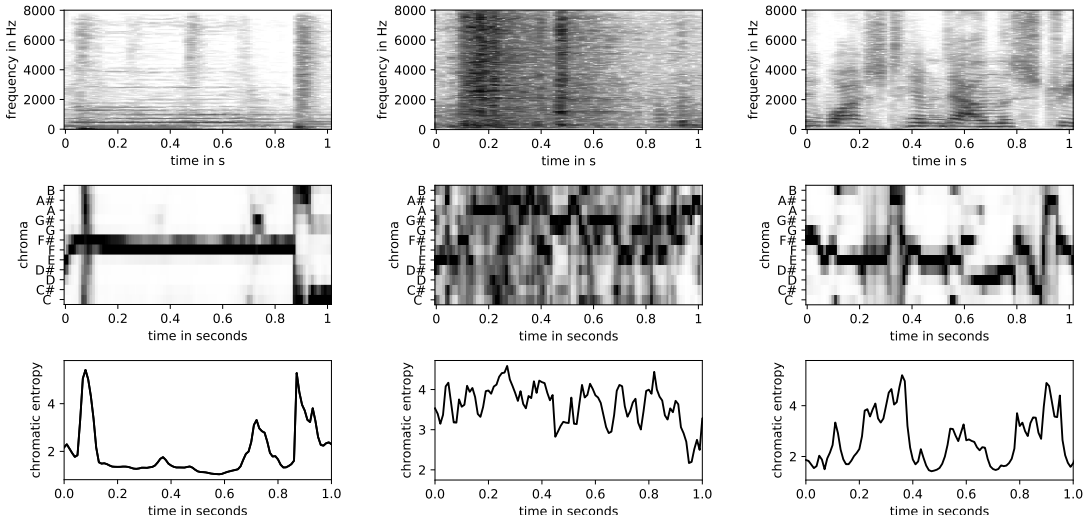
$$H_C[t] = -\sum_{k=1}^{12} p_k[t] \log(p_k[t]) \tag{5.16}$$

with normalised Chroma-features

$$p_k[t] = \frac{C[k,t]}{\sum_{i=1}^{12} C[i,t]}. \tag{5.17}$$

and chromagram $C[k,t]$ estimated at each time instance t and chroma k.

Figure 5.5 shows the chromagram $C[k,t]$ and the corresponding entropy H_C for music, speech and environmental noise signals respectively. As expected H_C is generally lower and more stationary for music (a) compared to speech (c). The entropy is considerably higher and fluctuates more rapidly for (unpitched) environmental noise.



(a) music (b) env. noise (c) speech

Figure 5.5 – Chroma Features and Chromatic Entropy for (a) music, (b) environmental noise and (c) speech.

Mel-scaling the spectrum

The Mel scale is a psychoacoustic scale of pitch perception, modeling the frequency response of the human auditory system: Human hearing is more sensitive in the lower frequency range where most speech and music content is located.

The Mel-scale describes the ability to discriminate pitch at different frequencies. At lower frequencies, very small increments in frequency can be discriminated. With rising frequency, the noticeable difference in pitch increases [96].

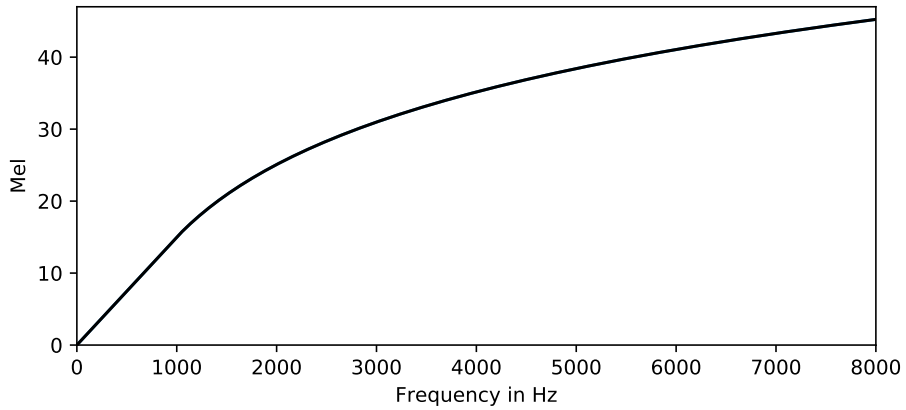


Figure 5.6 – Mel-scale according to Slaney's Auditory Toolbox [97], mapping the frequency-scale between 0 and 8kHz to Mel.

Slaney's definition of the Mel scale [97] was used for feature extraction, as implemented in librosa [86]. According to Slaney's implementation, the auditory system's frequency response is linear below $f_c = 1000$ Hz. Upwards of f_c, the

frequency resolution decreases incrementally. Frequency f can be converted to Mel as follows:

$$\text{Mel} = \begin{cases} \frac{f}{f_{sp}} & \text{for } f < f_c, \\ \frac{f_c}{f_{sp}} + \log\left(\frac{f}{f_c}\right) \frac{27}{\log(6.4)} & \text{otherwise,} \end{cases}$$

with spacing frequency $f_{sp} \approx 66.67$ (see figure 5.6).

Combining Linearly Spaced Frequency Bins to Mel Bins

N linearly spaced frequency bins k can be transformed to M Mel bins by weighted summation:

$$X_{\text{Mel}}[m] = \sum_{k=1}^{N} |X[k]| \cdot W[m,k], \tag{5.18}$$

with linearly spaced spectrum $X[k]$ and weights $W[m,k]$. $W[m,k]$ represents a filterbank with M triangular filters, as shown in figure 5.7. The filters are scaled so that the area under each triangle is constant.

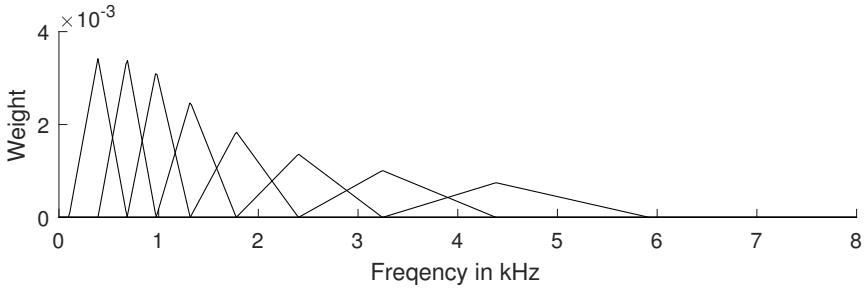
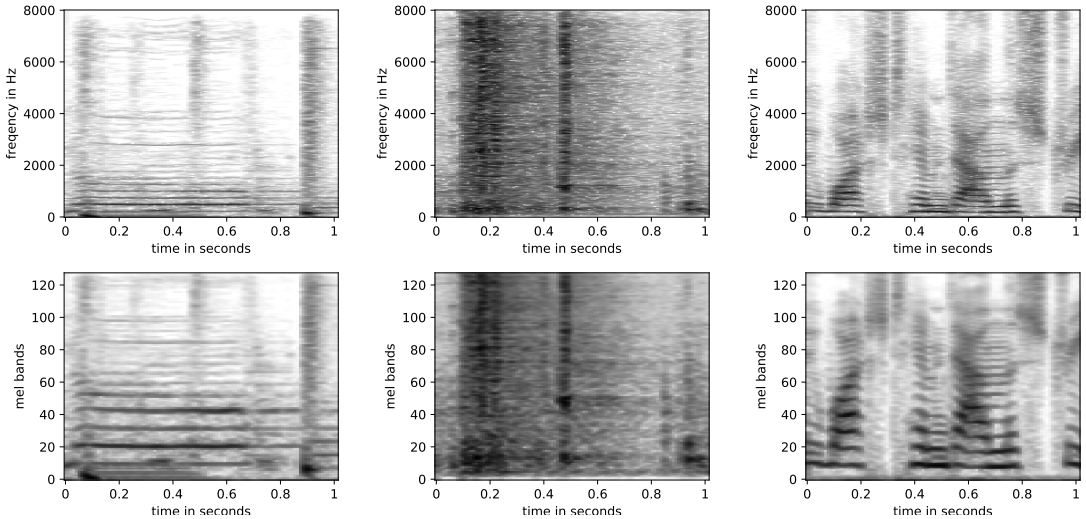


Figure 5.7 – Mel-weighting filter bank consisting of 8 triangular filters combing linearly spaced frequency bins between 0 and 8kHz into 8 Mel bands from 100Hz to 6kHz.

Figure 5.8 shows the spectrogram $|X[k,t]|_{\text{dB}}$ with 513 frequency bins linearly spaced between 0 and 8kHz and a Mel-spectrogram $|X_{\text{Mel}}[k,t]|_{\text{dB}}$ with 128 Mel bands for a music (a), environmental noise (b) and speech signal (c).

Characteristic patterns in the lower order harmonics of the music signal (a) (straight horizontal lines) and speech signal (b) (rising and falling lines interrupted by pauses and transients) are still easily observable. As can be seen, the relevant information is preserved while reducing the number of spectral components by 75% (513→128).



(a) music (b) env. noise (c) speech

Figure 5.8 – FFT spectrogram $|X[k,t]|_{\text{dB}}$ and Mel-spectrogram $|X_{\text{Mel}}[k,t]|_{\text{dB}}$ with 128 Mel bands for (a) music, (b) environmental noise and (c) speech.

5.1.1.3 Cepstral Features

Cepstral analysis (also quefrency analysis) was first introduced by Bogert et al. [98]. The terms cepstrum and quefrency were derived by reversing the first syllables of the terms spectrum and frequency. Interchanging the consonants symbolises the way of working in the frequency domain with methods, usually applied to time series and vice-versa [99].

Mel Frequency Cepstral Coefficients (MFCC)

The so-called mel frequency cepstral coefficients (MFCC) allow to accurately describe the shape of a signal's spectral envelope using only a few coefficients, making them popular features for various audio processing tasks.

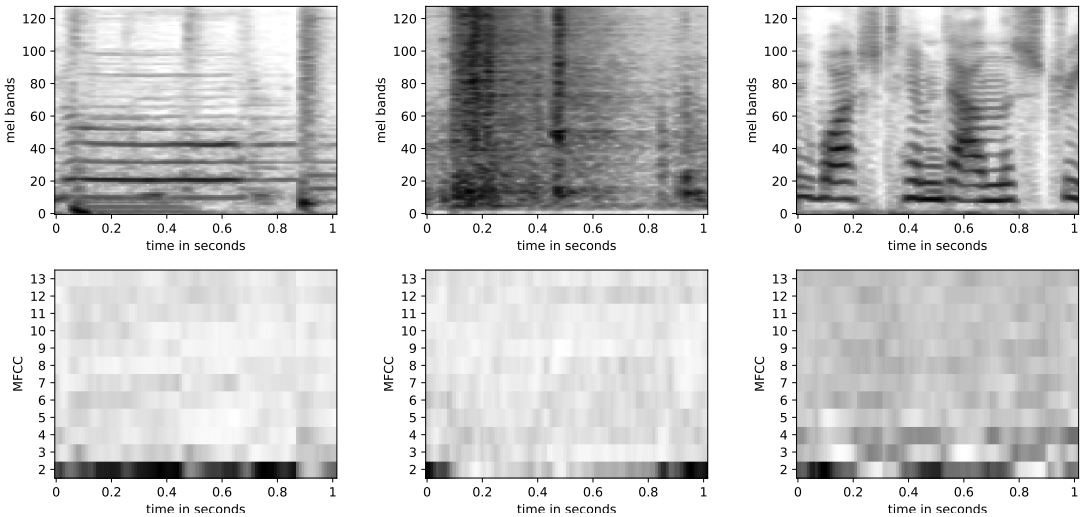
MFCCs are obtained by calculating the Discrete Cosine Transform (DCT) [5.1] of the logarithmic Mel-scaled magnitude spectrogram.

$$c_i[t] = \sqrt{\frac{2}{M}} \sum_{j=1}^{M} \log\left(X_{Mel}[k,t]\right) \cdot \cos\left(\frac{(i-1)\pi}{M}(k-0.5)\right) \qquad \forall\, i = 1,2,\ldots \tag{5.19}$$

where M represents the number of Mel filter banks [100]. Typically, only the first dozen MFCCs are evaluated. The first Coefficient $c_1[t]$ is equivalent to the signal energy and is discarded.

Figure 5.9 shows the MFCCs $c_i[t]$ with $i = 2,3,\ldots,13$ for audio clips of one second for the categories music, environmental noise and speech.

5.1. The Discrete Cosine Transform is similar to the Discrete Fourier Transform (DFT) but only uses the cosine function to represent the signal.



(a) music (b) env. noise (c) speech

Figure 5.9 – 12 first (omitting energy coefficient c_1) Mel-Frequency Cepstral Coefficients (MFCCs) for (a) music, (b) environmental noise and (c) speech.

5.1.1.4 Features from Harmonic Partials

As described in chapter 2, musical tones, voiced speech segments and some pitched environmental noise sounds have a distinct pitch, that is described by the fundamental frequency f_0. The sounds timbre is described by the prominence of the harmonics, i.e. multiples of the fundamental frequency.

Fundamental Frequency

The fundamental frequency f_0 describes the pitch of a sound. The *Sawtooth Waveform Inspired Pitch Estimator* (SWIPE) developed by Camacho and Harris [101] was used, as part of the *Speech Processing Toolkit* (SPTK) [87]. In essence, SWIPE estimates the pitch as the fundamental frequency of the sawtooth waveform that best matches the input signal. The comparison is done by computing a normalised inner product between the spectrum of the input signal and a kernel, representing the spectrum of a sawtooth signal.

Harmonicity

The harmonicity feature describes the ratio between the energy of the harmonic content and the total signal energy. It is higher for 'harmonically rich' sounds, with prominent harmonics and zero for unpitched sounds.

The harmonicity feature is defined as the ratio of harmonic energy $E_H[t]$ and total energy $E_T[t]$.

$$\text{Harm}[t] = \frac{E_H[t]}{E_T[t]}, \qquad E_H[t] = \sum_{h=1}^{H} a_h^2[t]. \tag{5.20}$$

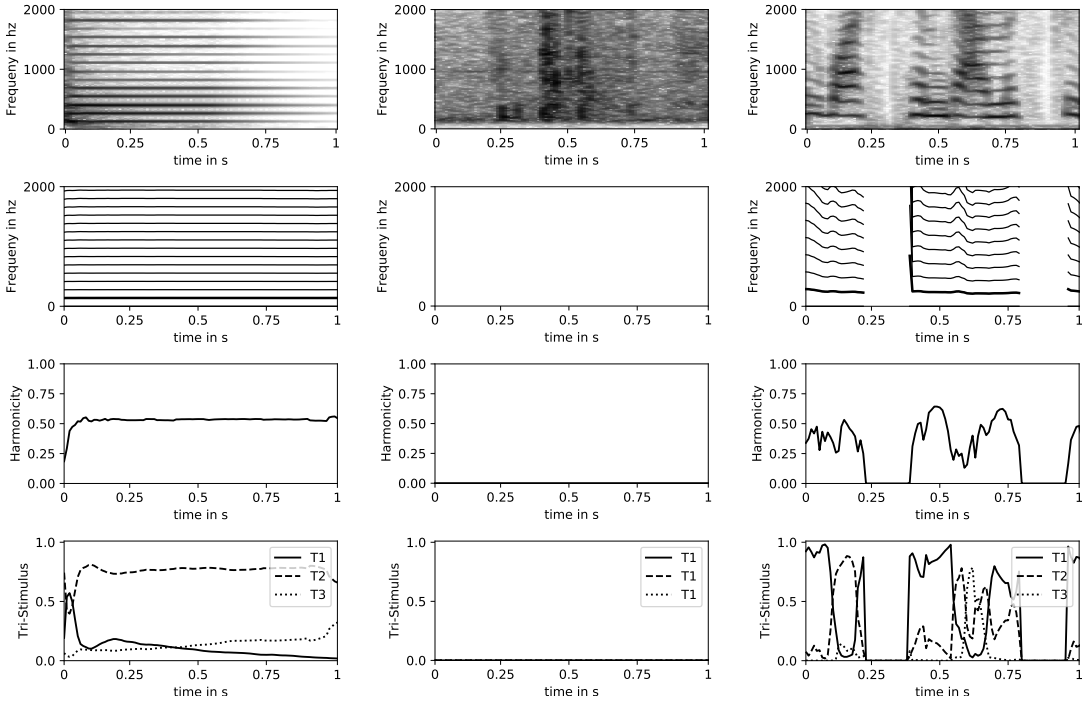
In accordance with [88] the amplitude of each harmonic partial $h = 1, \ldots, H$ is denoted as $a_h(t)$. A total of up to $H = 20$ harmonics are evaluated.

Tri-stimulus

The *Tri-stimulus* is set a timbral descriptors, describing the time-dependent behavior of musical transients. It was developed by Pollard and Jansson [102] as an equivalent of color attributes in vision.

The three stimuli T_1, T_2 and T_3 describe the mixture of harmonics, analogous to the three primary colors in vision (e.g. red, green and blue).

$$T_1 = \frac{a_1^2(t)}{\sum_{k=1}^{N} a_k^2(t)} \qquad T_2 = \frac{\sum_{h=2}^{4} a_h^2(t)}{\sum_{k=1}^{N} a_k^2(t)} \qquad T_3 = \frac{\sum_{h=5}^{H} a_h^2(t)}{\sum_{k=1}^{N}(t)} \tag{5.21}$$



(a) music (b) env. noise (c) speech

Figure 5.10 – Fundamental Frequency and (up to 20) harmonics, Harmonicity feature and tristimulus T_1, T_2, T_3 for (a) music, (b) environmental noise and (c) speech.

Figure 5.10 shows the estimated fundamental frequency f_0 of a speech signal, and its lower order harmonics, as well as the Harmonicity and Tri-stimulus features.

5.1.2 Clip-level Features - Temporal Integration

As mentioned before, it makes sense to integrate frame-level features over time into what is referred to as clip-level features [85].

One clip combines $N = 100$ overlapping frames, corresponding to one second of audio [5.2]. In literature, often longer clip lengths of 2s-3s are used [9–11]. However, to allow for a responsive real-time classification, it was chosen to use a clip-length of 1s (as suggested by [15, 24]).

Frame-level features can be combined to clip features in numerous ways. The two most common being mean and standard deviation of feature values over time.

L. Vrysis et al. [103] have reviewed the performance of several methods of temporal integration and found that besides mean and variance, considerable performance improvements in classification tasks have been achieved using different aggregated features, such as the Mean Absolute Sequential Difference (MASD), Low Crest Factor (LCF) and Relative Standard Deviation (RSD) of frame-level features.

For this thesis, the following temporal feature integration (TFI) procedures are evaluated:

Mean and Standard Deviation
The mean μ and standard deviation σ of attribute $a[n]$ are given as:

$$\mu = \frac{1}{N}\sum_{n=1}^{N} a[n], \qquad \sigma = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(a[n]-\mu)^2}, \tag{5.22}$$

for $n = 1, 2, \ldots, 100$ and a clip length of $N = 100$ time frames (equivalent of 1s).

Relative Standard Deviation
The *Relative Standard Deviation* (RSD) is the ratio of standard deviation σ and mean μ of attribute $a[n]$.

$$\text{RSD} = \frac{\sigma}{\mu} \tag{5.23}$$

Skewness and Kurtosis
The third and fourth statistical moments *Skewness* and *Kurtosis* of attribute $a[n]$ are given as:

$$\text{Skewness} = \frac{\frac{1}{N}\sum_{n=1}^{N}(a[n]-\mu)^3}{\sigma^3}, \qquad \text{Kurtosis} = \frac{\frac{1}{N}\sum_{n=1}^{N}(a[n]-\mu)^4}{\sigma^4}. \tag{5.24}$$

5.2. As described in chapter 5.1.1, features are evaluated 100 times per second (hop size of 10ms).

High and Low Crest Factor

High Crest Factor (HCF) and *Low Crest Factor* (LCF) are the ratios of the maximum or minimum value within one feature clip and the mean μ.

$$\text{HCF} = \frac{\max(\{a[1], a[2], \ldots, a[n]\})}{\mu}, \qquad \text{LCF} = \frac{\min(\{a[1], a[2], \ldots, a[n]\})}{\mu}. \tag{5.25}$$

Mean Sequential Difference

The *Mean Absolute Sequential Difference* (MASD) and *Mean Squared Sequential Difference* (MSSD) describe the amount of variation within each feature clip, similar to the standard deviation ω but with regard to the temporal evolution attribute $a[n]$.

$$\text{MASD} = \frac{1}{N-2}\sum_{n=2}^{N}|a[n]-a[n-1]|, \qquad \text{MSSD} = \frac{1}{N-2}\sum_{n=2}^{N}|x[n]-x[n-1]|^2. \tag{5.26}$$

Mean Crossing Rate

Similar to the Zero-Crossing Rate (see formula 5.1), the *Mean Crossing Rate* (MCR) estimates how often an attribute $a[n]$ crosses its mean value μ in a clip.

$$\text{MCR} = \sum_{n=1}^{N}\left(1-\delta(\text{sign}(a[n-1]-\mu), \text{sign}(a[n]-\mu))\right), \tag{5.27}$$

Frames Below Mean

Inspired by the *Percentage of Low Energy Frames* (%LEF) feature [11], a *Frames Below Mean* (FBM) temporal integration method is evaluated, counting the number of frames below than the mean μ.

$$\text{FBM} = \frac{1}{N}\sum_{n=1}^{N} g(a[n], \mu), \qquad \text{with} \qquad g(a,b) = \begin{cases} 1 & \text{for } a < b, \\ 0 & \text{otherwise.} \end{cases} \tag{5.28}$$

Note that besides MASD and MSSD, the temporal evolution of frame-level features is not considered and should be investigated in future work.

5.1.3 Feature Ranking and Selection

Evaluating 11 TFI methods for 26 frame-level features, yields a total of 286 descriptors. In order to obtain a more manageable feature set, features were ranked using different weighting algorithms. To do so, all 286 clip-level features were evaluated for 10800 non-overlapping audio clips, corresponding to 3h of audio data randomly drawn from the training dataset (see chapter 4), containing equal parts of music, speech and environmental noise. Prior to feature selection, attributes were standardised by subtracting their mean values and dividing by their standard deviation.

The attributes were then ranked using different feature weighting algorithms implemented in *Rapid Miner Studio* [5.3], namely weighting by Gini index [104], weighting by Information gain [105] and weighing by RELIEF [106].

The three feature ranking algorithms described above were then combined to obtain an average rank $\tilde{R}_a$ for each attribute a:

$$\tilde{R_a} = \frac{1}{3} \sum_{i=1}^{3} R_{i,a}, \tag{5.29}$$

where $R_{i,a}$ denotes the feature rank for method $i = 1, 2, 3$ and attribute a.

The 30 highest and 10 lowest ranked features for combined feature rank $\tilde{R_a}$, as well as their ranks and scores accoring to the Gini Index R_G, information gain R_{IG} and Relief R_R are listed in table 5.2.

Note that this feature selection procedure is very basic and was chosen to quickly obtain a manageable set of 30 descriptors. Other more sophisticated feature extraction algorithms should be investigated in the future.

As can be seen, top performing features are the standard deviation and MSSD of lower order MFCCs, indicating that the amount of timbral variation in an audio clip is a good descriptor for speech, music and environmental noise classification. Other highly ranked features include different temporal integrations of the Short-Time-Energy (STE), spectral moments and harmonicity feature.

The top 30 best-performing features were selected for classification using the classification algorithms described below.

5.3. *Rapid Miner Studio is a software for data science experiments, incorporating a wide variety of machine learning algorithms.* $https://rapidminer.com/products/studio/$ *- accessed: January 2018*

Rank	**Feature**	G	R_{G}	IG	R_{IG}	Relief	R_{R}	$\tilde{R}$
1	std_mfccs_02	0.2089	1	0.5294	1	0.5514	4	2.00
2	std_mfccs_00	0.1986	3	0.4917	3	0.4917	9	5.00
3	std_mfccs_01	0.1896	9	0.4871	4	0.4981	7	6.67
4	std_mfccs_03	0.1985	4	0.4966	2	0.3928	16	7.33
5	std_ste	0.2039	2	0.4726	9	0.4314	12	7.67
6	std_mfccs_04	0.1924	6	0.4868	5	0.4144	14	8.33
7	std_mfccs_06	0.1915	7	0.4821	7	0.3833	17	10.33
8	lcf_ste	0.1765	18	0.4119	19	1.0000	1	12.67
9	mean_ste	0.1805	16	0.4000	23	0.7049	2	13.67
10	mssd_mfccs_04	0.1904	8	0.4668	12	0.3138	28	16.00
11	std_mfccs_05	0.1866	12	0.4807	8	0.2958	32	17.33
12	mssd_mfccs_00	0.1878	10	0.4691	11	0.2553	43	21.33
13	mssd_mfccs_01	0.1809	15	0.4586	15	0.2827	38	22.67
14	mssd_mfccs_06	0.1847	14	0.4614	14	0.2543	44	24.00
15	lcf_spec_centroid	0.1440	33	0.3751	29	0.4801	11	24.33
16	std_spec_spread	0.1482	29	0.3928	25	0.3616	19	24.33
17	mssd_mfccs_03	0.1931	5	0.4846	6	0.2074	63	24.67
18	mssd_mfccs_02	0.1858	13	0.4720	10	0.2417	53	25.33
19	mssd_mfccs_05	0.1871	11	0.4644	13	0.2250	56	26.67
20	fbm_harmonicity	0.1424	35	0.3599	37	0.4847	10	27.33
21	std_mfccs_07	0.1699	20	0.4300	17	0.2534	46	27.67
22	std_mfccs_08	0.1599	21	0.4180	18	0.2443	52	30.33
23	lcf_spec_spread	0.1368	46	0.3629	35	0.3783	18	33.00
24	mssd_ste	0.1743	19	0.3946	24	0.2102	62	35.00
25	std_spec_rolloff	0.1377	45	0.3563	39	0.3457	22	35.33
26	mssd_mfccs_08	0.1572	23	0.3789	28	0.2233	57	36.00
27	lcf_spec_kurt	0.1268	56	0.3265	52	0.5075	6	38.00
28	mean_harmonicity	0.1444	31	0.3643	34	0.2451	50	38.33
29	rsd_spec_spread	0.1510	25	0.4069	20	0.1936	70	38.33
30	lcf_spec_rolloff	0.1258	57	0.3187	54	0.5311	5	38.67
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
277	skew_mfccs_10	0.0040	276	0.0089	276	0.0185	242	264.67
278	skew_mfccs_11	0.0042	275	0.0093	275	0.0181	244	264.67
279	skew_mfccs_09	0.0024	284	0.0049	284	0.0219	238	268.67
280	rsd_tristimulus_00	0.0060	261	0.0129	263	0.0000	284	269.33
281	lcf_harmonicity	0.0044	272	0.0106	273	0.0041	266	270.33
282	lcf_tristimulus_00	0.0045	270	0.0110	267	0.0013	274	270.33
283	skew_mfccs_07	0.0027	283	0.0062	282	0.0112	248	271.00
284	lcf_tristimulus_01	0.0045	269	0.0108	269	0.0011	277	271.67
285	lcf_tristimulus_02	0.0044	273	0.0107	272	0.0013	273	272.67
286	rsd_spec_rolloff	0.0001	286	0.0001	286	0.0000	286	286.00

Table 5.2 – 30 highest and 10 lowest ranked features based on average rank $\tilde{R}$.

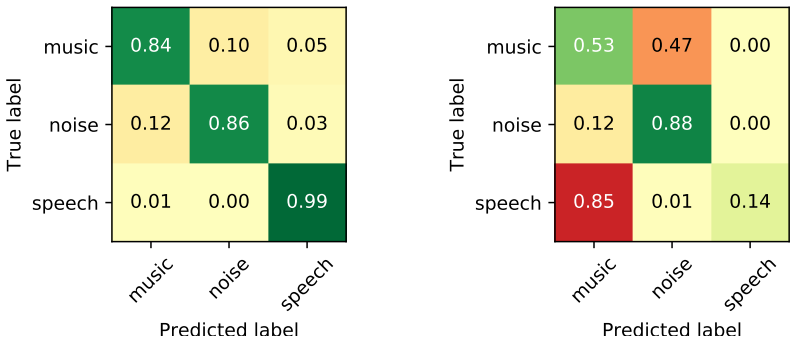
5.2 Naive Bayes Classifier

The Gaussian naive bayes (NB) classifier, as described in chapter 3.1.1 and implemented as part of the *scikit-learn* toolbox [107] for *Python* was trained and evaluated with the top 30 ranked clip features (see chapter 5.1.3) derived from data drawn from the MUSPEN dataset (see chapter 4).

The classifier does not have any open parameters and is used as a baseline, to compare other classifiers against.

5.2.1 Training with 'Clean' Audio Data

First, the classifier was trained using 75,600 clip-level feature vectors, derived from audio data randomly drawn from the *'clean'* training dataset, corresponding to 7h of audio data per class or 21h in total.



(a) test *'clean'* (b) test *'target'*

Figure 5.11 – confusion matrices for speech, music and environmental noise classification for naive bayes classifier trained with *'clean'* audio data and tested with (a) *'clean'* test data, and (b) *'target'* test data.

class	music	noise	speech	avg.
recall	84%	86%	99%	90%
precision	87%	90%	93%	90%
f-score	85%	88%	96%	90%

(a) test *'clean'*

music	noise	speech	avg.
53%	88%	14%	52%
35%	65%	100%	67%
42%	75%	25%	47%

(b) test *'target'*

Table 5.3 – naive bayes classifier trained with *'clean'* audio data and evaluated with *'clean'* and *'target'* test data respectively. Table shows precision, recall and f-score derived from the confusion matrix.

The trained model was then evaluated using the full *'clean'* test dataset, yielding a classification accuracy of approximately 90%. Figure 5.11 (a) shows the corresponding confusion matrix. Table 5.3 (a) lists the recall, precision and f-score

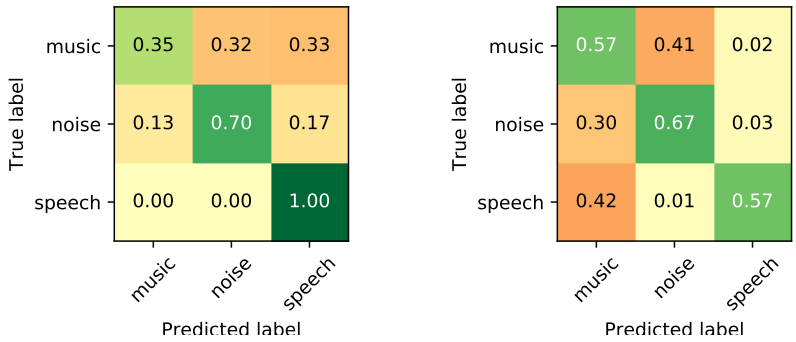
derived from the confusion matrix. As can be seen, speech data is almost always detected as such, with a recall of 99%. The discrimination between music and environmental noise is less successful with a recall of 84% and 86% respectively.

While the result is far from perfect, it shows that the underlying feature set contains enough information to discriminate speech from noise and music even with a simple classification algorithm like NB, at least for *'clean'* audio data.

However, when applying the classifier trained with *'clean'* audio data to real-world data, i.e. microphone array recordings, the overall classification accuracy drops to just 52%. The corresponding confusion matrix is shown in figure 5.11 (b). Speech is misclassified as music most of the times (85%) and music is often mislabelled as environmental noise (47%). The poor performance for the mismatched case (f-score of 47% vs. 90% for the matched case) indicates that the training data does not resemble the *'target'* domain well, leading to poor performance in real-world applications.

5.2.2 Training with 'Degraded' Audio Data

To overcome the mismatch between the *'target'* domain, i.e. real-world recordings with the far-field microphone array and the training data, the NB classifier was trained with the so-called *'degraded'* training dataset (see chapter 4.5). As with the *'clean'* training data (see above), the Gaussian NB classifier was trained using 75,600 feature vectors, derived from audio clips of 1s, randomly drawn from the *'degraded'* training set, corresponding to 21h of audio.



(a) Test Clean (b) Test Target

Figure 5.12 – confusion matrices for speech, music and environmental noise classification for naive bayes classifier trained with *'degraded'* audio data and tested with (a) *'clean'* test data, and (b) *'target'* test data.

Again, the trained model was evaluated using the *'clean'* test data, as well as the *'target'* microphone-array recordings. Figure 5.12 shows the corresponding confusion matrices. For *'clean'* test data, the classification accuracy significantly drops to approximately 68% (-22%) compared to training with *'clean'* data (see

above). Speech is always detected as such, however, music is only correctly recognised in one out of three cases.

class	music	noise	speech	avg.
recall	35%	70%	100%	68%
precision	73%	69%	67%	70%
f-score	47%	69%	80%	65%

(a) Test *'clean'*

music	noise	speech	avg.
57%	67%	57%	60%
44%	61%	92%	66%
50%	64%	70%	61%

(b) Test *'target'*

Table 5.4 – Naive Bayes classifier trained with *'degraded'* audio data and evaluated with *'clean'* test data and the *'target'* test data respectively. Table shows precision, recall and f-score derived from the confusion matrix.

The prediction accuracy for the *'target'* test set has improved considerably to 60% (+8%). The overall f-score increased greatly by 14% to around 61% (see table 5.4 (b)), indicating that augmenting the training data by simulating different acoustic environments helps to increase a classifier's performance in real-life application.

5.3 k-Nearest Neighbours classifier

As described in chapter 3, the kNN classifier discriminates new observation based on the class-membership of the k closest samples in the training dataset. The kNN algorithm as implemented in the scikit-learn toolbox for python [107] was used to classify the top 30 ranked clip-level features (section 5.1.3) with $k = 3$.

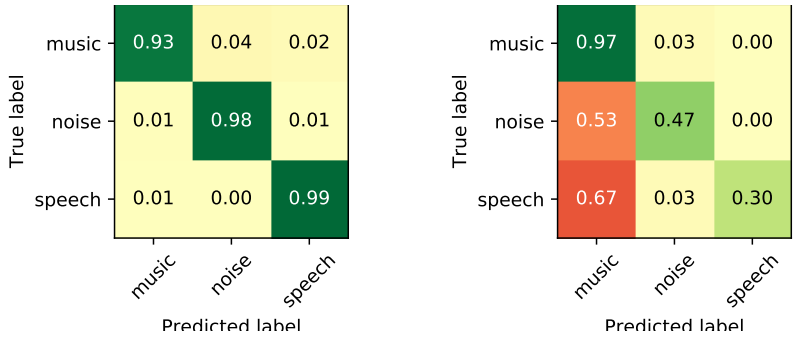
5.3.1 Training with 'Clean' Audio Data

First, the classifier was trained using 75,600 feature vectors, derived from audio clips of 1s, randomly drawn from the *'clean'* training dataset, corresponding to 7h of audio data per class or 21h in total. The trained model was then evaluated with the full *'clean'* and *'target'* test data. Figure 5.13 shows the corresponding confusion matrices. Table 5.5 shows the performance measures derived from the confusion matrices.

For *'clean'* test data, the kNN classifier performs very well with a classification accuracy of around 97%. Especially speech and environmental noise are detected reliably with a recall of 99% and 98% respectively. Considering the fact that there might be mislabelled data in the test set, this can be regarded as almost human-level performance.

However, when evaluating with *'target'* test data, classification accuracy drops to approximately 58% which is only slightly better than with the Gaussian naive bayes classifier (52%, section 5.2). Speech is often mislabelled as music (67%)

and environmental noise is often mislabelled as music (53%) resulting in a low f-score of only 56%.



(a) test *'clean'* (b) test *'target'*

Figure 5.13 – confusion matrices for speech, music and environmental noise classification for kNN ($k = 3$) classifier trained with *'clean'* audio data and tested with (a) *'clean'* test data and (b) *'target'* data.

class	music	noise	speech	avg.
recall	93%	98%	99%	97%
precision	98%	96%	97%	97%
f-score	95%	97%	98%	97%

(a) test *'clean'*

music	noise	speech	avg.
97%	47%	30%	58%
45%	89%	100%	78%
61%	62%	46%	56%

(b) test *'target'*

Table 5.5 – kNN classifier trained with *'clean'* audio data and evaluated with *'clean'* test data and the *'target'* test data respectively. Table shows precision, recall and f-score derived from the confusion matrix.

5.3.2 Training with 'Degraded' Audio Data

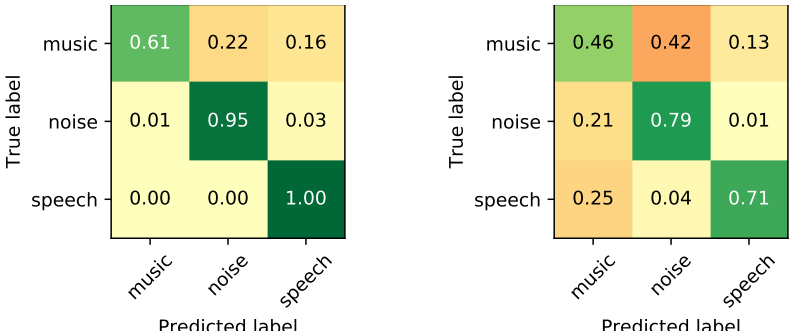
To overcome the mismatch between the *target* domain, i.e. real-world recordings with the far-field microphone array and the training data, the kNN classifier was then trained with the so-called *'degraded'* training dataset (see chapter 4.5). The classifier was trained using 75,600 feature vectors, derived from audio clips of 1s, randomly drawn from the *'degraded'* training set, corresponding to 21h of audio.

Again, the trained model was evaluated using *'clean'* test data, as well as the *'target'* microphone-array recordings. Figure 5.12 shows the corresponding confusion matrices. Table 5.6 shows the performance measures derived from the confusion matrices.

For *'clean'* test data, the classification accuracy is around 85% (compared to 68% with the NB classifier, section 5.2). Especially environmental noise and speech are

detected reliably. However, music is quite often mislabelled as env. noise (22%) and speech (16%), yielding an f-score of 84%.

For the *'target'* test data, performance is significantly improved compared to training with *'clean'* audio data (f-score of 65% vs. 56%, see tables 5.6 and 5.5). Also the kNN algorithm is more reliable than the NB classifier trained with *'degraded'* audio data (f-score of 65% vs 61%, see tables 5.6 and 5.4). However, the overall performance for the *'target'* domain is still quite poor.



(a) test *'clean'* (b) test *'target'*

Figure 5.14 – confusion matrices for kNN ($k = 3$) classifier trained with *'degraded'* audio data and tested with (a) *'clean'* and (b) *'target'* test data.

class	music	noise	speech	avg.
recall	61%	95%	100%	85%
precision	98%	81%	84%	88%
f-score	75%	87%	91%	84%

(a) test *'clean'*

music	noise	speech	avg.
46%	79%	71%	65%
50%	63%	84%	66%
48%	70%	77%	65%

(b) test *'target'*

Table 5.6 – kNN classifier trained with *'degraded'* audio data and evaluated with *'clean'* test data and the *'target'* test data respectively. Table shows precision, recall and f-score derived from the confusion matrix.

5.4 Support Vector Machine

The support vector machine (SVM) as implemented in the *scikit-learn* toolbox [107] for *Python* was used. A soft-margin SVM with non-linear radial basis function (RBF) kernel was used, with open parameters γ and C (see chapter 3.1.3). The top 30 ranked clip-level features (see section 5.1.2) were evaluated.

The open parameters γ and C were chosen based on a parameter grid search algorithm.

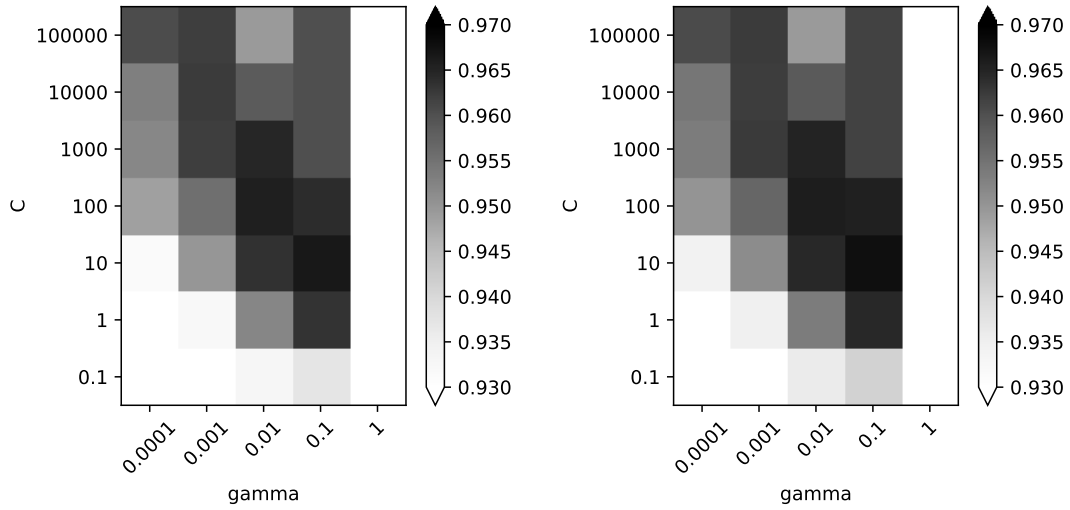
Coarse-level parameter optimisation
A grid-search algorithm with stratified 10-fold cross-validation [5.4] is used to find well-performing parameters for γ and C. First a coarse parameter grid is evaluated with parameter values for γ logarithmically spaced between 10^{-4} and 1, and values for C ranging from 10^{-1} to 10^{5} respectively. A one-vs-rest (OVR) multi-class implementation is used.

Parameter optimisation is done using a subset of 3,600 audio clips, corresponding to 1h of audio randomly drawn from the *Clean* training dataset (see chapter 4). The best performance was achieved using penalty parameter $C = 1$ and kernel parameter $\gamma = 0.1$ for both precision and recall, as can be seen in figure 5.15, showing the average validation scores for precision (left) and recall (right) along the investigated parameter grid.

Fine-tuning parameters
For fine-tuning the parameters, a logarithmic grid was evaluated with $\gamma \in [0.01, 1]$ and $C \in [1, 100]$. Figure 5.16 shows the parameter space drawn by parameters γ and C. best parameters for precision were $\gamma \approx 0.042$ and $C \approx 11.72$, best parameters for recall were $\gamma = 0.069$ and $C \approx 1.74$.

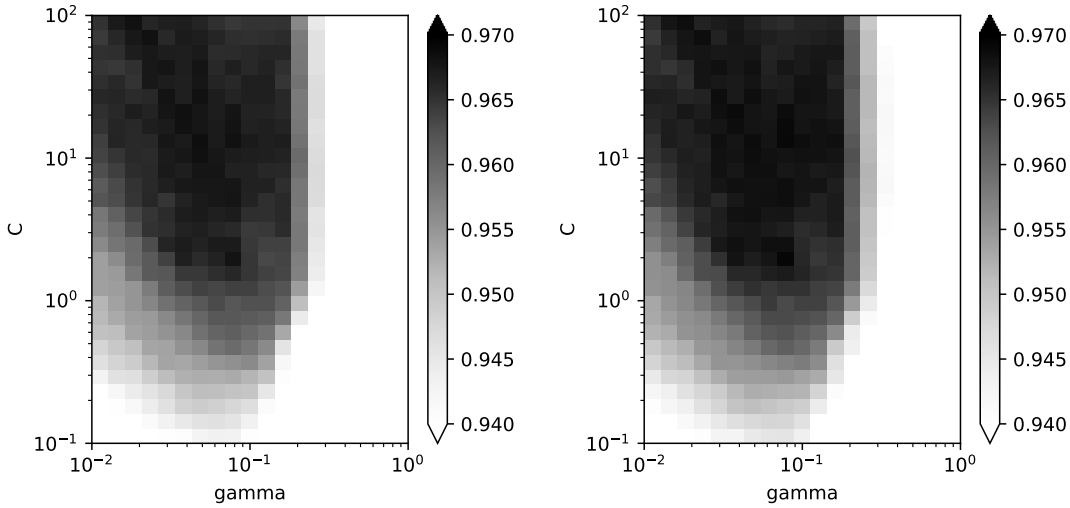
Based on the fine-level grid evaluation, parameters $C = 5$ and $\gamma = 0.05$ are chosen.

5.4. k-fold cross-validation splits the training data into k subsets. The model is evaluated k times. Each time one of the k subsets is used for testing. The other $(k-1)$ subsets are used for training. The average validation scores are then computed for all k evaluations. In stratified k-fold cross-validation, the percentage of samples for each class is preserved for each subset.



(a) precision (b) recall

Figure 5.15 – Tuning parameters for support vector machine with RBF kernel. Optimising 10-fold cross-validation score for (a) precision and (b) recall. Darker spots indicate better performance.



(a) precision (b) recall

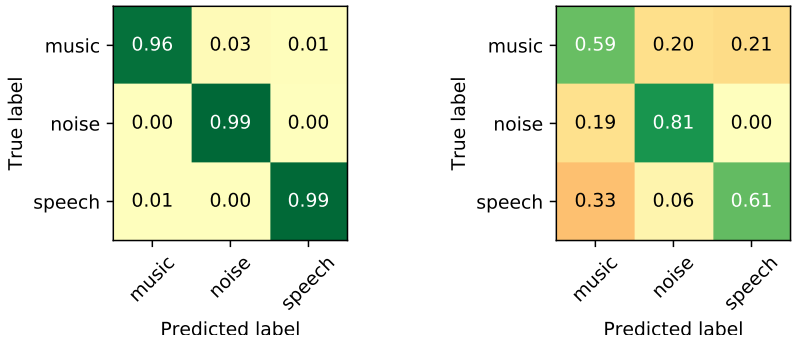
Figure 5.16 – Fine-tuning parameters for support vector machine with RBF kernel. Optimising 10-fold cross-validation score for (a) precision and (b) recall. Darker color indicates better performance.

5.4.1 Training with 'Clean' Audio Data

The SVM with RBF kernel is trained with clip-level features derived from 21h of *'clean'* training data randomly drawn from the raw audio data (see chapter 4) with equal parts music, speech and environmental noise. Table 5.7 lists the classification performance for the different evaluation sets. Figure 5.17 shows the corresponding confusion matrices.

For *'clean'* test data, an impressive accuracy of approximately 98% is achieved. Given the fact that the test data might contain mislabelled data, this can be seen as at least human-level performance. Especially environmental noise and speech are detected very reliably with a recall of 99%. Music is sometimes mislabelled as environmental noise (3%).

However, as with the other classification algorithms, when evaluating the classifier with the *'target'* test data, performance drops drastically, with a classification accuracy of only 67%.



(a) test *'clean'* (c) test *'target'*

Figure 5.17 – confusion matrices for SVM with RBF kernel trained with *'clean'* audio data and tested with (a) *'clean'* test and (b) *'target'* test data.

class	music	noise	speech	avg.
recall	96%	99%	99%	98%
precision	99%	97%	99%	98%
f-score	97%	98%	99%	98%

(a) test *'clean'*

music	noise	speech	avg.
59%	81%	61%	67%
53%	76%	74%	68%
56%	78%	67%	67%

(b) test *'target'*

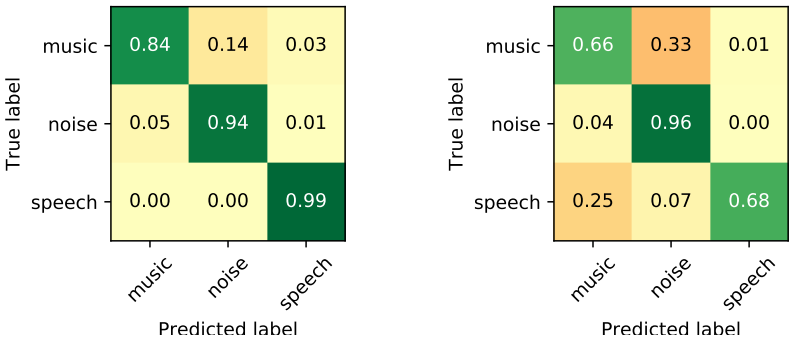
Table 5.7 – SVM classifier trained with *'clean'* audio data and evaluated with *'clean'* test data and the *'target'* test data respectively. Table shows precision, recall and f-score derived from the confusion matrix.

5.4.2 Training with 'Degraded' Audio Data

In order to reduce the mismatch between the training data and the *'target'* domain, i.e. real-world microphone array recordings, the SVM with RBF kernel is trained with clip-level features derived from 21h of audio randomly drawn from the *'degraded'* audio data (see chapter 4.5). Figure 5.17 and table 5.7 show the corresponding confusion matrices and derived performance measures.

For *'clean'* test data, classification accuracy is approximately 92%. While this is significantly less, than when trained with *'clean'* audio data (-6%), environmental noise and speech are still detected reliably (recall of 94% and 99% respectively). Music is mislabelled as environmental noise quite often though (14%).

For *'target'* test data, i.e. microphone array recordings, classification accuracy improves to around 77% (+9%) compared to training with *'clean'* audio data (see tables 5.8 and 5.7). Even-though the overall performance for the *'target'* test data is much improved, music is often misclassified as noise (33%) and speech often misclassified as speech (25%), as can be seen in figure 5.18.



(a) test *'clean'* (b) test *'target'*

Figure 5.18 – confusion matrices for SVM with RBF kernel trained with *'degraded'* audio data and evaluated with (a) *'clean'* and (b) *'target'* test data.

class	music	noise	speech	avg.
recall	84%	94%	99%	92%
precision	94%	87%	96%	92%
f-score	89%	90%	97%	92%

(a) test *'clean'*

music	noise	speech	avg.
66%	96%	68%	77%
69%	71%	98%	79%
67%	82%	80%	76%

(b) test *'target'*

Table 5.8 – SVM classifier trained with *'degraded'* audio data and evaluated with *'clean'* test data and the *'target'* test data respectively. Table shows precision, recall and f-score derived from the confusion matrix.

5.5 Summary

As has been shown, traditional machine learning algorithms (namely naive bayes, kNN and SVM) have been used successfully to distinguish between speech, music and environmental noise for *'clean'* audio data, resembling broadcast audio. The clip-level features carry enough information to effectively discriminate between the three classes, at least for low-noise audio data. Accuracies of up to 98% were achieved for a SVM with RBF kernel.

However, the models trained with *'clean'* audio data, performed poorly for far-field microphone array recordings. In order to overcome the mismatch between the *'clean'* training data and the *'target'* domain, the models were also trained with the *'degraded'* audio data, simulating different acoustic environments (see chapter 4.5). While performance has been greatly improved, the resulting classifiers are still not capable of reliably discriminating speech and music from environmental noise in noisy and reverberant environments (i.e. *'target'* test data).

For future work, using a more reliable set of features might improve performance, but requires expert knowledge about the signal characteristics. Considering the performance gain achieved by using *'degraded'* audio data, augmenting the training dataset by actual far-field microphone recordings is expected to improve performance further.

6 | Deep Learning Approach: Convolutional Neural Nets

As described in chapter 1.2, deep learning algorithms have been applied to audio classification tasks with great success. Especially so-called convolutional neural nets (CNNs) have proven effective for classification tasks, such as speaker identification [29], audio-based action recognition [30] and large-scale acoustic event detection [32].

In contrast to the feature-based ML approaches described in chapter 5, deep learning algorithms don't require the definition and selection of descriptive features. Instead, a deep neural net will learn to extract relevant features from the raw input (or a simple mid-level signal representation) during training.

The underlying mechanics of CNNs have been explained in chapter 3.2. This chapter describes the deep learning approach with CNNs developed within the scope of this thesis, which is based on the neural net topology proposed by Lukic et al. [29].

6.1 Input Signal Representation: Mel-spectrograms

As investigated by Dieleman and Schrauwen [108], deep neural nets are capable of learning relevant features directly from raw audio data. However, while they achieved promising results for raw audio data, for CNNs superior performance was reported using a 2D input signal representation, i.e. Mel-spectrograms. Essentially, this transforms the classification of a time series $x[t]$ into an image classification problem.

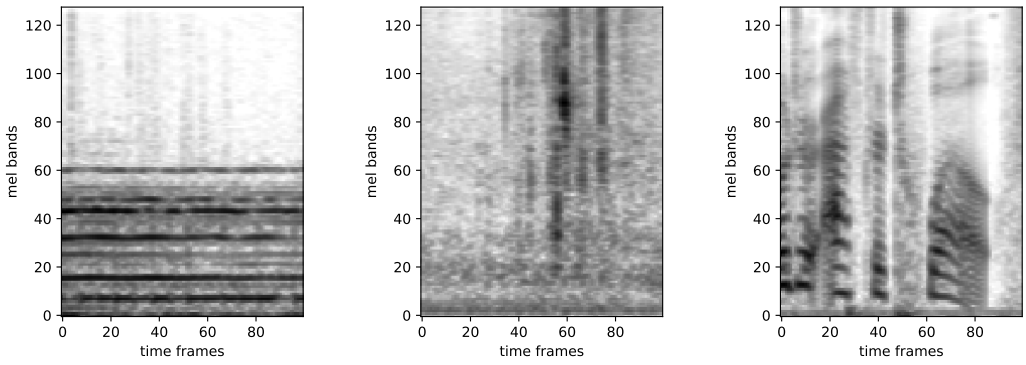
CNNs with two-dimensional convolutional layers aim to find patterns within an input image, by correlating small image-segments with learned filter kernels (see chapter 3.2.4).

The most obvious choice for a two-dimensional audio signal representation is the so-called spectrogram as described in chapter 5.1.1.2. As has been established in chapter 2, most characteristic signal components of speech and music signals

(i.e. fundamental frequency and lower order harmonics) are located in the lower frequency range, where the human ear is the most sensitive. It thus makes sense to use a tempo-spectral signal representation modelling the non-linearity of the human auditory system's frequency resolution, such as the Mel-spectrogram (see chapter 5.1.1.2).

Mel-spectrograms have been used in conjunction with CNNs with great success [29,32]. As proposed by Lukic et al. [29] and Hershey et al. [32] Mel-spectrograms were calculated for audio clips of 1s. A clip length of 1s has proven to be a good compromise between sufficient temporal integration and responsiveness. If frames shorter than 1s are used, there will be more non-characteristic frames, leading to misclassification. Longer segments will make real-time classification slow and unresponsive, as the prediction will depend on acoustic events that occurred several seconds ago.

Figure 6.1 shows Mel-spectrograms for 1s of music, environmental noise and speech respectively. As can be seen, the signal characteristics for (a) music and (c) speech, i.e. stationary horizontal lines for music and the succession of voiced and unvoiced speech segments - as described in chapter 2, can be easily observed. The environmental noise signal is more evenly spread across frequency and time with some transient events occurring around half way through the audio clip.



(a) music (b) env. noise (c) speech

Figure 6.1 – Mel-spectrograms for 1s of (a) music, (b) environmental noise and (c) speech.

Depending on the final application of the classification algorithm, other signal representations should be investigated. Initial experiments with the proposed CNN model (see section 6.2) suggest that using so-called cochleagrams [6.1] yields comparable results for the same number of spectral components.

6.1. Cochleagrams are an approximation to the filtering performed by the human ear. A filter-bank of rectangular band-pass filters models the human auditory system. Similar to the Mel-scale, the non-linearity in frequency resolution is modelled according to the so-called equivalent rectangular bandwidth (ERB) scale. The energy of each filter's output models the nervous activity at a specific location in the cochlea, a spiral-shaped cavity in the inner ear, containing the Organ of Corti, the sensory organ of auditory perception [3,109].

Time vs. Frequency resolution
If frequency-resolution is too low, it will be hard to detect small changes in pitch, which can carry useful information (e.g. prosody in speech signals). If time-resolution is too low, transients and rhythmic patterns will be 'washed out' and harder to detect.

Based on [29], Mel-spectrograms were computed from an FFT with a window size of 1024 samples (64ms), a hop size of 160 samples (10ms) for audio sampled at f_s = 16kHz, yielding 100 time frames for audio clips of 1s. The resulting 513 frequency bins between 0 and 8kHz were combined to 128 Mel bands (see chapter 5.1.1.2).

This yields an input image of 128×100 pixels [6.2].

Dynamic Range Compression
Instead of using the linear magnitude of the spectrogram, usually some form of non-linear amplitude compression is applied when training convolutional neural nets on spectrograms, as this type of non-linearity can be hard to learn for a neural net with only ReLu activation functions (see chapter 3.2.1) [108]. For this thesis, the logarithmic power spectrum (as described in chapter 5.1.1.2) and the resulting Mel-spectrogram in dB is used.

Contrast Normalisation
The logarithmic magnitude of each input image $\mathbf{X}$ is scaled between 0 and 1 by first deducting the minimum of each sample and consecutive normalisation (division by maximum). This alleviates the problem of varying input signal levels (e.g. when the source is closer or further away from the microphone) without the need for automatic gain control (AGC).

$$\mathbf{X} \leftarrow \mathbf{X} - \min(\mathbf{X}), \qquad \mathbf{X} \leftarrow \frac{\mathbf{X}}{\max(\mathbf{X})} \tag{6.1}$$

6.2. In order to scale down the CNN for embedded applications, it might be necessary to reduce the resolution of the Mel-spectrograms. Initial tests indicate that Mel-spectrograms with at least 64 Mel bands and 50 time frames per second yield good results using the proposed CNN model. Mel-spectrograms with 24 Mel bands and 32 time frames per second still yielded recognition rates of around 90% for *'clean'* test data, indicating that the model can be scaled down without too much of a hit on performance.

6.2 Model structure

The basic model structure is based on the CNN model proposed by Lukic et al. [29] for speaker identification and clustering. The model consists of two convolutional layers, each with subsequent max pooling and two fully-connected layers. Rectifier linear units (ReLu) are used for all layers except the output layer, for which the softmax activation was used. Dropout was evaluated between the two fully-connected layers.

Figure 6.2 shows the basic model structure of the investigated CNN. Two convolutional layers with subsequent pooling layers extract relevant information from the input image. Two fully connected layers and an output layer form an MLP classifier, which outputs the estimated a-posterior probability for each class.

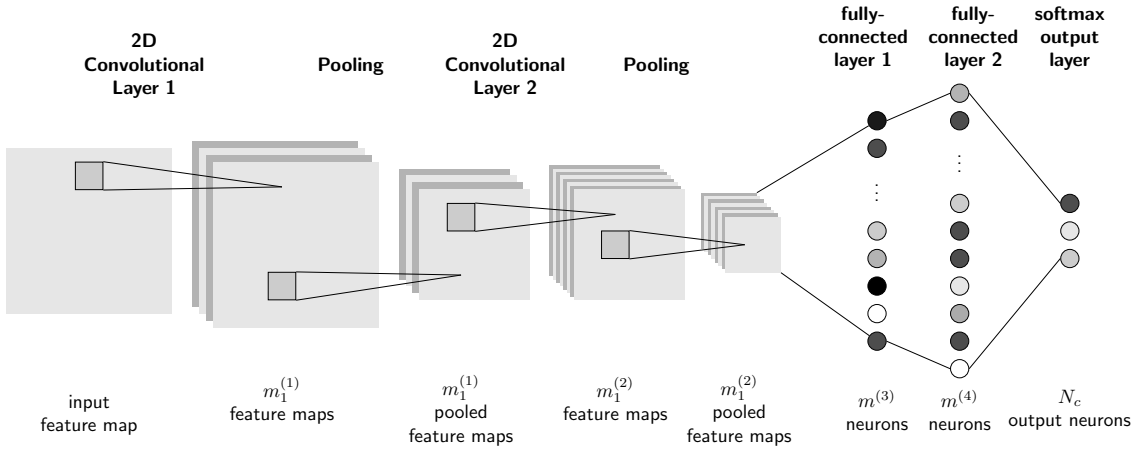


Figure 6.2 – Model structure of CNN featuring two convolutional layers with consecutive pooling and an MLP classifier with two fully-connected layers for image classification.

Number of Convolutional Layers

Deep learning breaks the analysis of raw sensory input data into a series of simple mappings. The so-called hidden layers extract increasingly abstract features from the input data. The first convolutional layer in an image classifier will learn simple features like edges in various directions, by comparing the brightness of neighbouring data points (pixels) in the input image (in our case a Mel-spectrogram). The second convolutional layer will learn more complex contours. With rising depth of the DNN, input representations get more and more complex [45, p.6].

CNNs for image classification, like AlexNet [27] or GoogLeNet [110] are usually many layers deep, as they have to differentiate very complex objects. For this thesis CNNs with only two convolutional layers are evaluated, as the input images are less complex.

Fundamentally, the system needs to be able to extract horizontal lines (for tonal music segments), rising and falling lines (fundamental frequency and harmonics of voiced speech), vertical lines (transient signals, like drum sounds or transient noise events) and local noise. Using only one convolutional layer severely decreased the models performance in initial tests and was not further investigated.

Kernel Sizes

Lukic et al. [29] have studied kernels of different sizes $h_1 \times h_2$ along the frequency and time axis. They found that good results can be achieved using both symmetrical and asymmetrical kernels, concluding that two-dimensional filter kernels make the model more easily interpretable. They settle for 4×4 kernels with strides 1×1. For this thesis, greater 8×8 kernels were investigated as well but did not increase performance (see chapter 6.4).

As suggested in [29], the resulting feature maps were pooled using a pool size 4×4 and stride 2×2.

Number of Filters

As mentioned before, kernels of the first and second convolutional layers represent simple visual features, such as edges and contours in the spectrogram. Visualising convolutional filters (see chapter 6.7 below) can help, finding the right number of filters. When training a CNN with many filters for each convolutional layer, there might be quite a few that look very similar, indicating that the number of filters can be reduced, without a significant loss in performance.

If the model is too complex, overfitting will be more likely. If there are too few filters, on the other hand, the model might not be able to extract relevant information from the input image.

As mentioned above, the CNN needs to effectively differentiate harmonic content (horizontal lines) and transient events (vertical lines) in order to differentiate between music, speech and environmental noise segments. For the speaker identification proposed by Lukic et al. [29], the differentiation of the spectrograms needs to be more precise. It was thus chosen to use less than the 32 and 64 filters for the first and second convolutional layer used by Lukic et al.. Instead, 8 and 16 filters were evaluated for the first, 8, 16 and 32 for the second convolutional layer.

Fully-connected Layers

The convolutional layers with consecutive pooling layers are followed by two fully-connected layers (also called dense layers) and an output layer, forming an MLP classifier [6.3].

If the number of neurons is too low, the neural net will struggle to locate learned patterns (filter kernels) along the frequency and time axis. If the number of neurons

6.3. Feed-forward neural nets with two-hidden layers are known to perform excellent in classification tasks and usually better than those with only one hidden layer [111].

is too high, the model becomes overly complex as the number of trainable parameters (i.e. interconnection weights) rises at least exponentially with increasing number of neurons.

Lukic et al. [29] suggest to chose the number of neurons depending on the number of classes to predict ($10 \cdot N_c$ for the first and $5 \cdot N_c$ for the second dense layer) and a dropout rate of 50% between the two fully connected layers.

For this thesis 32, 64 and 128 computing units are evaluated for the first fully-connected layer, 16, 32 and 64 for the second fully-connected layer.

Activation Functions
ReLu activation functions were used for all layers except the output layer. Instead, the softmax activation function (equation 3.28 in chapter 3.2.1) was used for the output neurons, so that the output vector $\tilde{\mathbf{y}} = [\tilde{y}^{(1)}, \tilde{y}^{(2)}, \tilde{y}^{(3)}]^T$ of the CNN estimates the a-posteriori probabilities for each of the three classes.

When using a sigmoid activation function instead, the system can theoretically detect mixtures of classes [32]. Initial experiments indicate that this could be promising for future work, especially when differentiating more classes and subcategories.

Initialisation
In accordance with Glorot and Bengio [112] all biases were initialised with 0. Interconnection weights and convolutional filter kernels were initialised with the common heuristic

$$\mathbf{W}^{(l)} \sim U\left[-\frac{1}{\sqrt{m^{(l-1)}}}, \frac{1}{\sqrt{m^{(l-1)}}}\right], \tag{6.2}$$

where $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$ and $m^{(l-1)}$ denotes the number of neurons in the previous layer $(l-1)$.

6.3 Sanity Checks

Before training a neural net with a lot of data, which might take hours or even days, several sanity checks can be performed to make sure training will go as intended.

Loss before training
First of all, to see if the model is correctly initialised and does not suffer from an inherent bias, the untrained network should yield a prediction accuracy of $a = 1/N_c$, where N_c is the number of classes.

For speech, music and environmental noise discrimination, prediction accuracy a should be approximately 33%. The corresponding loss (categorical cross-entropy, see chapter 3.2.3) is $\mathcal{L} = -\log(\frac{1}{N}) = -\log(\frac{1}{3}) \approx 1.098$.

Overfitting a small subset of data
To see if the model is capable of discriminating the training data, a small subset of data is over-fitted, so that an accuracy of $a = 100\%$ is achieved when evaluating the network with the same data. This ensures that the network is able to extract any relevant information from the input data.

Figure 6.3 shows the training loss for the a neural net trained and evaluated with $n = 1200$ Mel-spectrograms per class, computed from audio clips randomly drawn from the full MUSPEN dataset (see chapter 4), corresponding to one hour in total. As expected, the validation loss starts at $\mathcal{L} \approx 1.098$ and rapidly drops to zero.

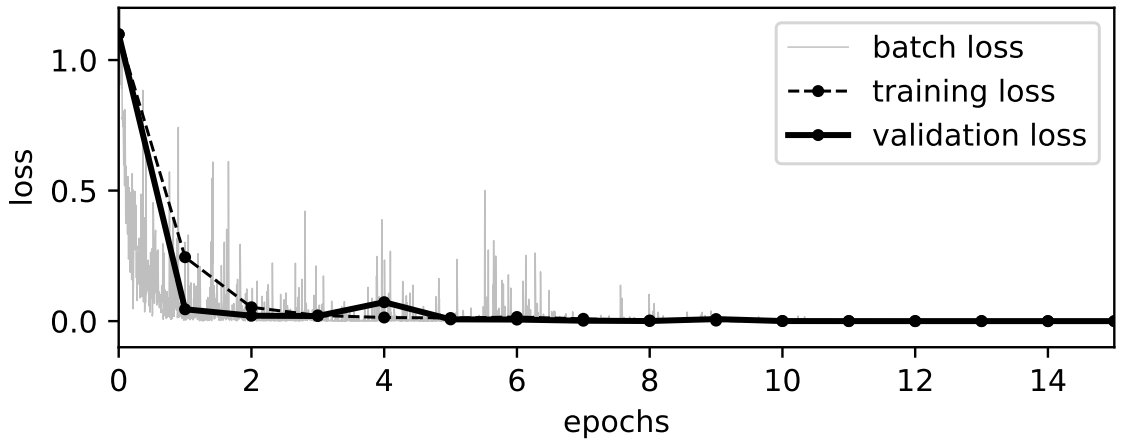


Figure 6.3 – Training progress for overfitting experiment. A deep neural net was trained for 15 epochs and evaluated with the full training data after each epoch.

6.4 Choosing Hyperparameters

Complex neural net topologies like CNNs have many tunable hyperparameters, such as the number of layers, activation functions, number of filters, etc..

Hyperparameters can either be chosen manually or automatically using a grid search algorithm. The manual approach requires a profound knowledge of what each parameter does, and comes down to intuition and experience. Automatic hyperparamter selection on the other hand does not require as much expert knowledge but is computationally costly [45, p.422].

As explained above, the basic model structure was chosen as it has been proven successful for audio classification before [29]. As the classification task in speech, music and environmental noise is expected to require less features than for speaker identification, the number of convolutional filters was reduced.

In order to find a model structure that works well for speech, music and environmental noise classification, a total of 216 different hyperparameter settings was evaluated, as listed in table 6.1.

Layer:			**# of settings**
Conv. Layer 1:	kernels	4×4, 8×8	2
	strides	1×1	1
	filters	8, 16	2
	activation	ReLu	1
Pooling Layer 1:	pools	4×4	1
	strides	2×2	1
Conv. Layer 2:	kernels	4×4, 8×8	2
	strides	1×1	1
	filters	8, 16, 32	3
	activation	ReLu	1
Pooling Layer 2:	pools	4×4	1
	strides	2×2	1
Dense Layer 1:	neurons	32, 64, 128	3
	activation	ReLu	1
Dropout:	dropout	0.5	1
Dense Layer 2:	neurons	16, 32, 64	3
	activation	ReLu	1
Output Layer:	neurons	3	1
	activation	Softmax	1
		# number of combinations: $\prod \rightarrow$	216

Table 6.1 – Evaluated hyperparameters for CNN model.

For each of the 216 different hyperparameter settings, the CNN was trained with 144,000 Mel-spectrograms per class, representing around 40h of training data randomly drawn from the full training data in the MUSPEN dataset, with equal parts of *'clean'* and *'degraded'* data (see chapter 4). Each CNN was trained for 10 epochs with a batch size of $B = 128$ and optimiser *Adam*. After each epoch the full *'target'* test set was evaluated. The best accuracy was kept to evaluate each hyperparameter setting.

Table 6.2 lists the average, maximum and minimum performance achieved for each hyperparameter within the grid of 216 parameter configurations. Note that even-though the grid search algorithm only investigates a small subset of possible hyperparameter configurations, it took several days to compute.

choosing hyperparameters for 1st convolutional layer:

# of filters:	8	16
avg. acc.	73%	73%
max. acc.	85%	84%
min. acc	61%	60%
stdv.	5%	5%

kernel:	4×4	8×8
avg. acc.	76%	71%
max. acc.	85%	81%
min. acc	63%	60%
stdv.	5%	4%

choosing hyperparameters for 2nd convolutional layer:

# of filters:	8	16	32
avg. acc.	72%	74%	74%
max. acc.	83%	85%	84%
min. acc	60%	61%	63%
stdv.	6%	5%	5%

kernel:	4×4	8×8
avg. acc.	75%	72%
max. acc.	85%	81%
min. acc	62%	60%
stdv.	5%	5%

choosing number of hidden units:

1st fully-connected layer

# of neurons:	32	64	128
avg. acc.	72%	74%	75%
max. acc.	82%	85%	85%
min. acc	62%	61%	60%
stdv.	5%	5%	5%

2nd fully-connected layer

# of neurons:	16	32	64
avg. acc.	73%	73%	74%
max. acc.	84%	83%	85%
min. acc	60%	61%	61%
stdv.	5%	5%	5%

Table 6.2 – How investigated hyperparameters affect performance: This table shows the average, maximum and minimum accuracy for each investigated hyperparameter within the 216 combinations evaluated.

As can be seen, when looking at the top section of table 6.2 the number of filter kernels in the first convolutional layer does not affect performance noticeably. However, using kernel sizes of 4×4 yielded better results than using larger kernels (+5%).

Increasing the number of filter kernels in the second convolutional layer beyond 16 did not increase performance with an average accuracy of 74%. As with the first convolutional layer, using smaller kernel sizes of 4×4 yielded better results than using larger kernels (+3%).

When looking at the lower section of table 6.2, one can see that increasing the number of hidden units in both the first and second fully-connected layer does slightly increase average accuracy (at the cost of exponentially more trainable parameters).

Based on the hyperparameter analysis, the following network hyperparameters, as listed in table 6.3, were chosen for further experiments.

Layer:			**output shape:**	**# of params.:**
Input Layer:	mel-spec.	128×100	128×100×1	0
Conv. Layer 1:	kernel	4×4		
	strides	1×1		
	filters	8, 16		$8 \cdot 4 \cdot 4 + 8$
	activation	ReLu	128×100×8	= 136
Pooling Layer 1:	pools	4×4		
	strides	2×2	64×50×8	0
Conv. Layer 2:	kernel	4×4		
	strides	1×1		
	filters	16		$8 \cdot 16 \cdot 4 \cdot 4 + 16$
	activation	ReLu	64×50×16	= 2,064
Pooling Layer 2:	pools	4×4		
	strides	2×2	32×25×16	0
Dense Layer 1:	neurons	128		$32 \cdot 25 \cdot 16 \cdot 128 + 128$
	activation	ReLu	128	= 1,638,528
Dropout:	dropout	0.5	128	0
Dense Layer 2:	neurons	64		$128 \cdot 64 + 64$
	activation	ReLu	64	= 8256
Output Layer:	neurons	3	3	$64 \cdot 3 + 3$
	activation	Softmax		= 195
				total: $\sum \rightarrow$ 1,649,179

Table 6.3 – Chosen hyperparameters for CNN model. Note that 99% of all trainable parameters are located in the first fully-connected layer.

As can be seen when looking at table 6.3, around 99% (~1.6M) of all trainable parameters are located in the first fully-connected layer. If computational resources are limited, the number of trainable parameters can be effectively reduced without much of a loss in performance when using less hidden units in the first and second fully-connected layers (see also table 6.2).

For example when using 64 instead of 128 hidden units in the first fully-connected layer, the number of trainable parameters decreases by almost 50% to 825,819 while performance only decreases by a few percent.

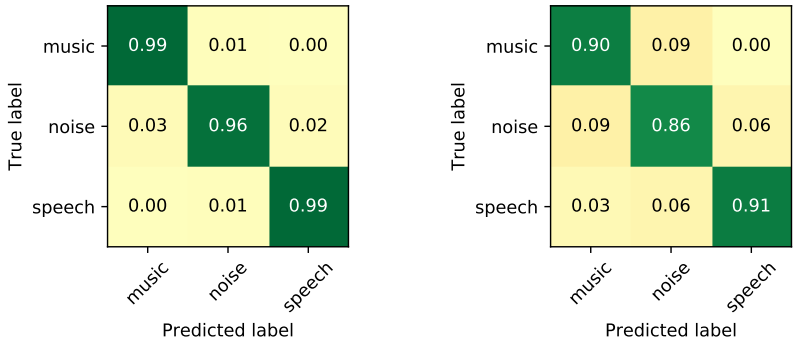
6.5 Offline Evaluation

In order to evaluate the performance of CNNs for speech, music and environmental noise classification, the CNN with hyperparameters listed in table 6.3 was trained with the full *'clean'* training data, as well as the full *'degraded'* training data.

In both cases, the CNN was trained for a maximum of 15 epochs. After each epoch, the full *'clean'* test data, as well as the full *'target'* test data were evaluated. The best performing weights for both the *'clean'* and *'target'* test data were stored. The CNN was trained using the categorical cross entropy as a loss function and *Adam* as the optimiser for batches of size $B = 128$. Training data was shuffled before each epoch.

6.5.1 Training with 'Clean' Audio Data

First the CNN was trained with the full *'clean'* training dataset (around one million Mel-spectrograms) and evaluated with the *'clean'* and *'target'* test data. Figure 6.4 and table 6.4 show the corresponding confusion matrices and derived performance measures for (a) *'clean'* test data and (b) the recorded dataset coming from the far-field microphone array.



(a) Test Clean (b) Test Target

Figure 6.4 – confusion matrices for CNN classifier trained with *'clean'* audio data and tested with (a) *'clean'* and (b) *'target'* test data.

As can be seen, the CNN trained with *'clean'* audio data does an excellent job in detecting speech and music for the *'clean'* test set, resembling broadcast audio. Music and speech are correctly detected with a recall of 99%. Environmental noise is sometimes falsely labelled as music (3%) and speech (2%) which is not surprising, given the inhomogeneous nature of environmental noise sounds. The overall classification accuracy is around 98%. When considering the fact that there might be mislabelled data in the test set, this can be regarded as human-level performance and is comparable to the SVM trained and evaluated with *'clean'* audio data (see chapter 5.4).

class	music	noise	speech	avg.
recall	99%	96%	99%	98%
precision	97%	98%	98%	98%
f-score	98%	97%	98%	98%

(a) Test *'clean'*

music	noise	speech	avg.
90%	86%	91%	89%
88%	85%	94%	89%
89%	85%	92%	89%

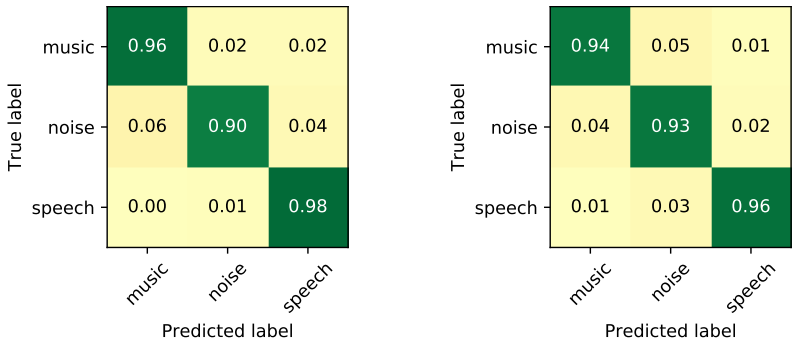
(b) Test *'target'*

Table 6.4 – CNN classifier trained with *'clean'* audio data and evaluated with *'clean'* test data and the *'target'* test data respectively. Table shows precision, recall and f-score derived from the confusion matrix.

Interestingly, the CNN trained with *'clean'* audio data works surprisingly well for the 'target' test data, with a classification accuracy of 89%, outperforming the feature-based approaches by a comfortable margin. This could also be verified in real-time tests (see chapter 6.8).

6.5.2 Training with 'Degraded' Audio Data

As with the feature-based classifiers, the model was then trained with the *'degraded'* dataset in order to reduce the mismatch between training data and the *'target'* domain.



(a) test *'clean'* (b) test *'target'*

Figure 6.5 – confusion matrices for CNN classification trained with *'degraded'* data and tested with (a) *'clean'* and (b) *'target'* test data.

The CNN was trained with the full *'degraded'* training dataset (around one million Mel-spectrograms) and evaluated again with both the *'clean'* and *'target'* test data.

Figure 6.5 shows the corresponding confusion matrices for (a) *'clean'* test data and (b) the recorded *'target'* data coming from the far-field microphone array. The classifier's performance is listed in table 6.5.

As can be seen when looking at figure 6.5 (a), the performance for the *'clean'* test data is still very good. However, around 10% of environmental noise data are

class	music	noise	speech	avg.
recall	96%	90%	98%	95%
precision	94%	97%	94%	95%
f-score	95%	93%	96%	95%

(a) test *'clean'*

music	noise	speech	avg.
94%	93%	96%	94%
95%	92%	97%	95%
94%	92%	96%	94%

(b) test *'target'*

Table 6.5 – CNN classifier trained with *'degraded'* audio data and evaluated with *'clean'* test data and the *'target'* test data respectively. Table shows precision, recall and f-score derived from the confusion matrix.

misclassified as either speech or music, compared to just under 5% when training with *'clean'* audio data. The overall accuracy decreases to approximately 95%.

More interestingly though, the overall accuracy for the *'target'* test data increases to approximately 94%. The CNN trained with degraded audio also performed well in real-time tests (see 6.8), allowing a responsive and robust detection of speech and music, even when the source is several metres away from the microphone array.

Overall the CNN proves to be a very robust classifier for audio data, even if heavily degraded by background noise and reverberation, outperforming other feature-based approaches. Note that the CNN has been trained with considerably more audio data than the feature-based algorithms and is substantially more computationally demanding. In contrast to the feature-based machine learning approach (see chapter 5), where expert knowledge is required to define and select relevant descriptive features, the deep neural net learns to extract relevant patterns from the input signal representation (i.e. Mel-spectrogram) during training. In consequence, the CNN is more robust and less prone to noise.

6.6 Common misclassifications

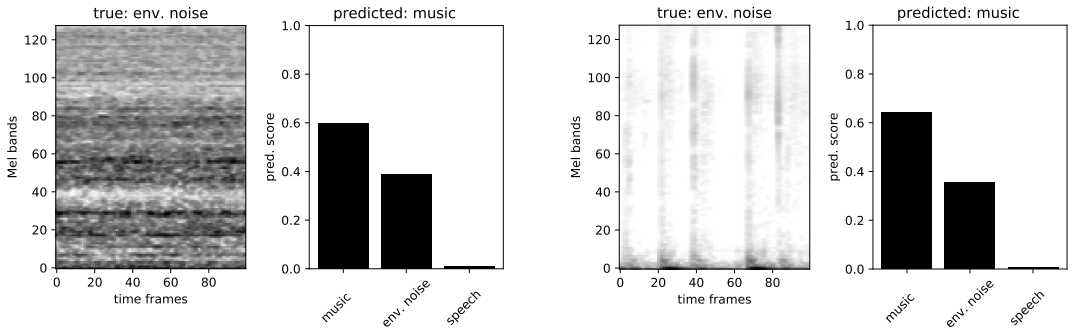
While classification accuracy is generally good, in some cases the CNN will mislabel audio segments. In the following, common misclassification for a CNN trained with *'degraded'* audio data are shown. The examples include both *'clean'* and *'target'* test data.

Environmental Noise misclassified as Music

Sometimes, environmental noise with prominent harmonic content will be misinterpreted as music, as shown in figure 6.6 (a). This is to be expected, as even humans sometimes can have a hard time differentiating tonal noise events from music, especially for short segments.

As shown in figure 6.6 (b), environmental noise sounds with a prominent rhythmic pattern (which might sound similar to a drum pattern) are sometimes mislabelled as music.

In both cases, the neural net is not certain about its decision, with predicted probabilities of about 60% for music and 40% for environmental noise.

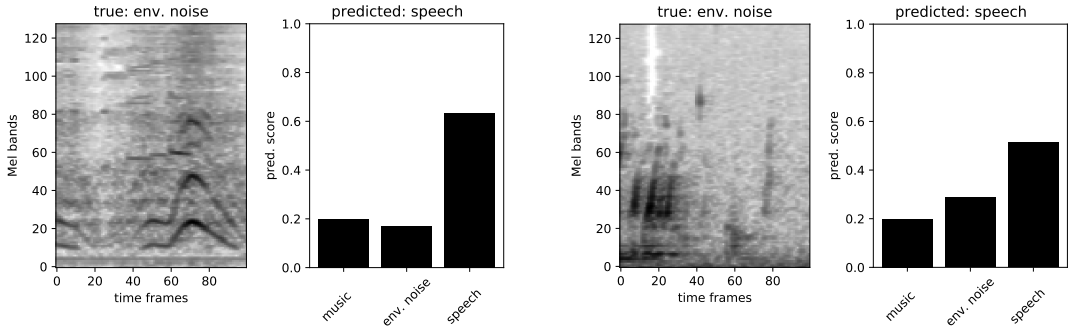


(a) pitched environmental noise sound (b) env. noise sound with rhythmic pattern

Figure 6.6 – Environmental noise segments mislabelled as music.

Environmental Noise misclassified as Speech

As described in chapter 2, speech spectrograms are characterised by their varying fundamental frequency and its lower order harmonics. In rare cases, such as depicted in figure 6.7 (a), environmental noise sounds with varying pitch can lead to misclassification. Figure 6.7 (b) shows another, although less characteristic environmental noise sound mislabelled as speech.

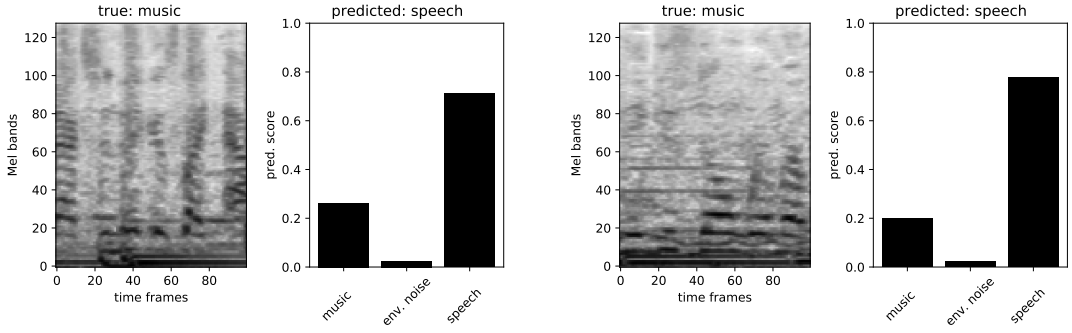


(a) env. noise sound with varying pitch. (b) env. noise sound with

Figure 6.7 – Environmental noise segments mislabelled as speech.

Music misclassified as Speech

Quite often music contains some sort of speech (see figure 6.8). Especially in rap music, as shown in (a), the differentiation between music and speech is often difficult and sometimes even impossible. The Mel-spectrogram inherits both characteristics of speech signals and music, leading to a false prediction.



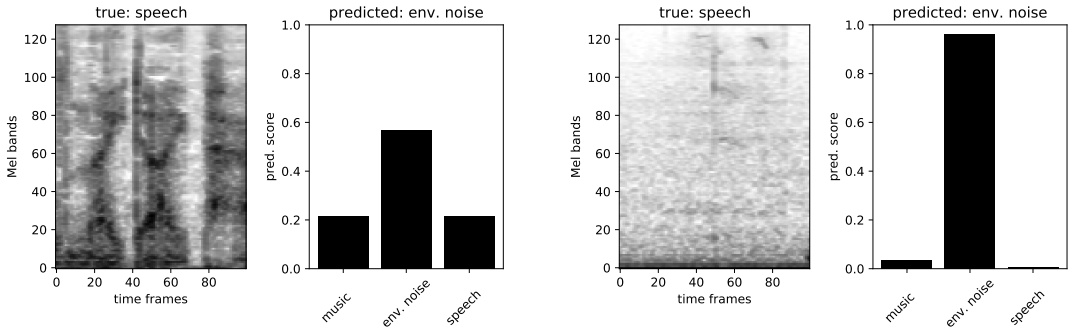
(a) rap music segment (b) music with singer

Figure 6.8 – Music segments mislabelled as speech.

Speech misclassified as Environmental Noise

If speech signals are too heavily distorted, the CNN will fail to identify them as such (figure 6.9 (a)).

Pauses are an essential part of speech. However, longer pauses in speech signals are not considered speech in this context. Even-though a silence-removal procedure was applied to speech training data, there appears to be some mislabelled data left in the test set (figure 6.9 (b)).



(a) heavily distorted speech signal (b) mislabelled data in test set

Figure 6.9 – Speech segments mislabelled as environmental noise.

Looking at mislabelled data in the test set provides some information on what kind of data is hard for the neural net to distinguish. However, it is not possible to say, *why* the classifier identifies a sample to be of a certain class. To gain an insight on the deciding factors of a neural net classifier, class models can be visualised, as described below.

6.7 Class Model Visualisation by Inverting the Neural Net

One drawback of using deep neural nets for classification is that the underlying decision rules of a neural net cannot be easily understood, sometimes making the results hard to interpret.

One approach to alleviate this problem has been introduced by Simonyan et al. [113] in 2013. They propose a method to visualise learned class models of a CNN by inverting the model structure and generating an input image that maximises the class score $s_C(\mathbf{I})$:

$$\underset{\mathbf{I}}{\operatorname{argmax}}\, s_c(\mathbf{I}) - \lambda \cdot ||\mathbf{I}||_2^2, \tag{6.3}$$

with a regularisation parameter λ. A synthetic input image $\mathbf{I}$ characteristic for class c is obtained by iteratively changing the pixels with regard to the activation of the classifier's output neurons [6.4]. More specifically, a stochastic gradient ascend is performed, maximising $s_c(\mathbf{I})$ with each step:

$$\mathbf{I} \leftarrow \mathbf{I} + \Delta\mathbf{I} \tag{6.4}$$

Similar to the gradient descent during training (see chapter 3.2), the image is adapted in the direction of the gradient $\nabla s_c(\mathbf{I})$ (steepest ascent). As suggested by [114], momentum was added to the gradient ascent.

$$\Delta\mathbf{I} \leftarrow \gamma \cdot \nabla s_c(\mathbf{I}) + m \cdot \Delta\mathbf{I} = \gamma \cdot \frac{\partial\, s_c(\mathbf{I})}{\partial\, \mathbf{I}} + m \cdot \Delta\mathbf{I} \tag{6.5}$$

with step size γ, gradient ∇s_c, and decaying factor m.

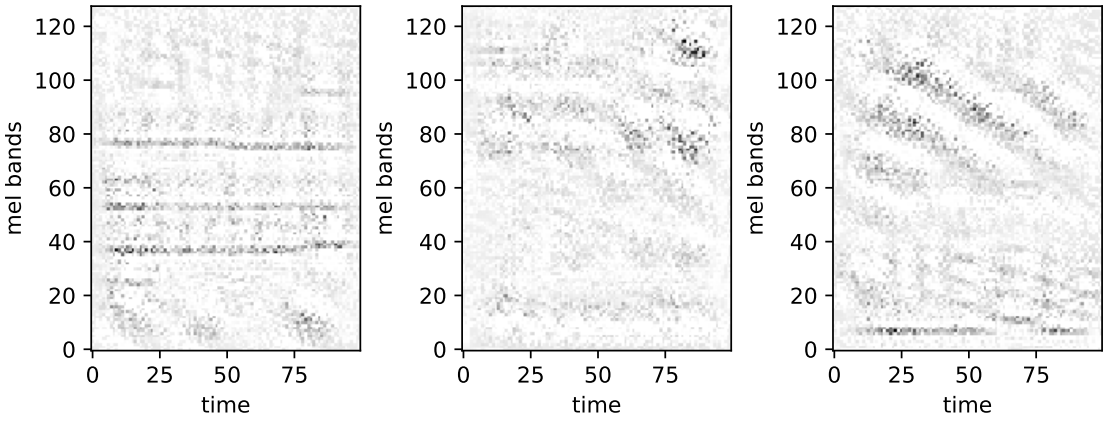
Figure 6.10 shows model visualisations (sometimes called deep dream images) for the classes music (a), environmental noise (b) and speech (c), coming from a CNN trained with Mel-spectrograms. The images represent synthetic Mel-spectrograms that result in a strong activation of the respective output neuron. This allows a certain insight into what signal characteristics are relevant for classification. However, the quality of the images and how well they resemble an actual Mel-spectrogram does not say much about the system's performance, and is largely dependent on the initial image (in our case random noise), chosen model parameters (such as kernel sizes, strides, ...), step size γ, decaying factor m and the number of iterations.

Looking at the images, one can see that the algorithm learns to identify music based on straight horizontal lines in the spectrogram. Note that even though this image suggests that the detection of music might only be dependant of the tonal structure of a musical excerpt, the trained model also works well on isolated drum sequences without harmonic / tonal content.

6.4. Note that the activations *before* the output layer's softmax activation function are evaluated to make sure that the optimisation is only dependent of the class c itself. Otherwise, the score s_c will be maximised by minimising the scores for the other classes [113].

Speech on the other hand is classified based the shape of the fundamental frequency and lower order harmonics, which can be observed when looking at the lower Mel bands in figure 6.10 (c). Additionally, temporal fluctuations of signal energy in the higher frequency range seem to be characteristic for speech signals (i.e. succession of consonants).

The generated image for environmental noise (b) does look less characteristic, which is not surprising given the inhomogeneous nature of environmental noise sounds. Interestingly it does not contain any horizontal (harmonic) structures in the lower and mid frequency range.



(a) Music (b) Env. Noise (c) Speech

Figure 6.10 – Class model visualisations for the three classes music (a), environmental noise (b) and music (c). CNN with two convolutional layers (kernel size 8×8 with stride 1×1), each with consecutive max pooling (pool size 4×4 with stride 2×2) and two fully connected layers with 64 and 32 neurons was trained with 1,728,000 Mel-spectrograms with 128 Mel bands and 100 time frames, corresponding to 1s of audio.

Convolutional Layer Kernel Visualisation

In a similar fashion, the activations of convolutional layer filters can be visualised, resulting in images that result in a strong activation for each learned filter kernel. The input image is iteratively changed to maximise the activation of a convolutional filter kernel. As stated above, this can help to see whether the training process is going into the right direction.

Figure 6.11 shows the visualisation of 16 filter kernels of the first convolutional layer after training for several epochs. Some kernels represent vertical and horizontal lines of different strengths. As quite a few of the resulting images look very similar (e.g. filters 5, 6 and 13 or filters 3 and 16), the number of filters can probably be reduced without a great hit on performance. Also two of the images do not show any characteristic patterns but random noise (filters 12 and 15). Ideally, all filter kernels show characteristic, unique and distinct patterns.

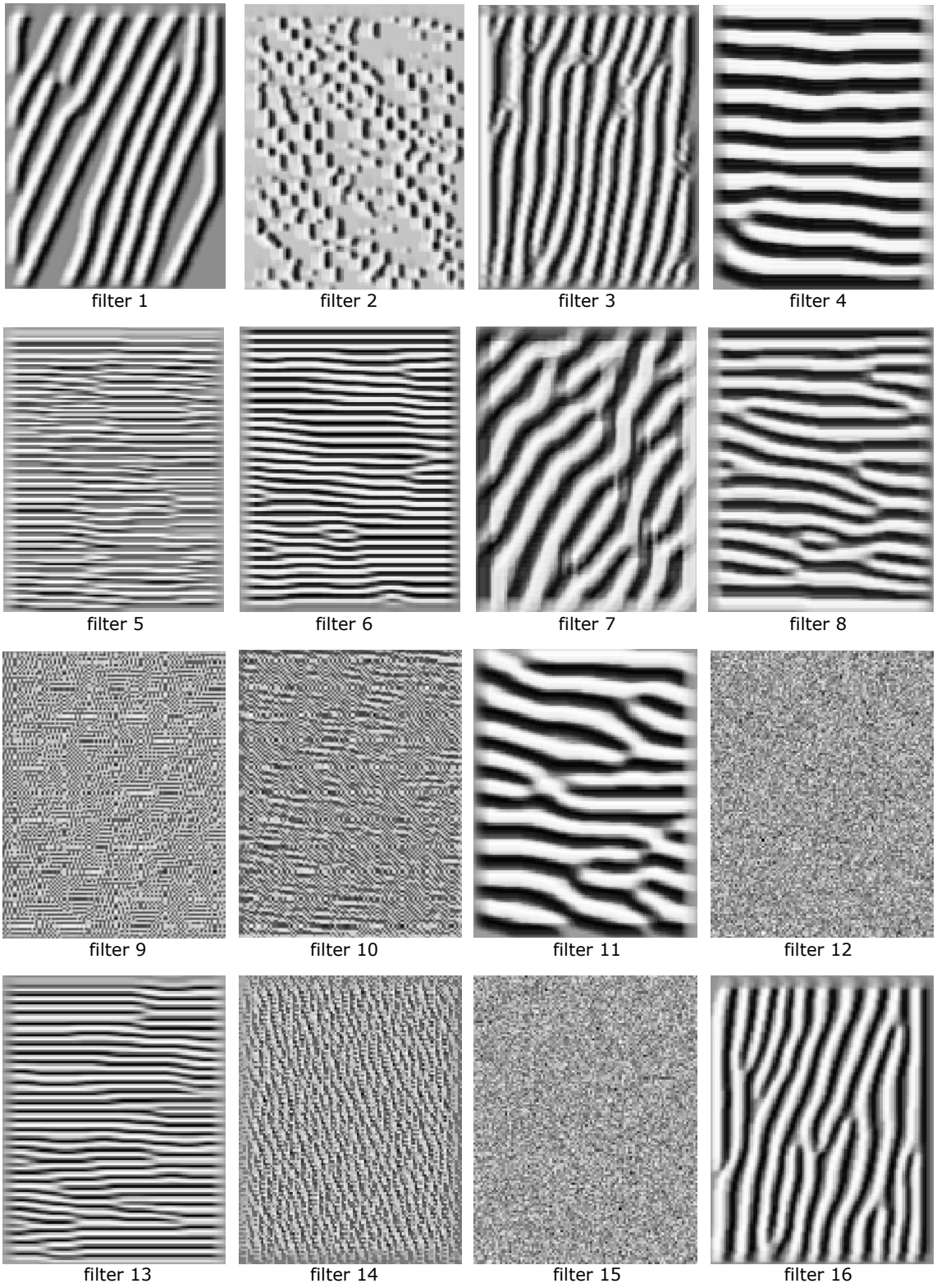


Figure 6.11 – Visualising learned filter kernels of first convolutional layer.

6.8 Real-time Implementation

In order to demonstrate the capabilities of the proposed CNN classifier, a *'proof-of-concept'* real-time classification procedure was implemented.

More specifically, audio data is streamed from the microphone array via a USB audio-interface [6.5] and buffered for one second using the pyAudio Toolkit [115]. The buffer is re-written and evaluated 12 times per second, meaning that a new prediction of class membership is obtained every ~83.33 ms.

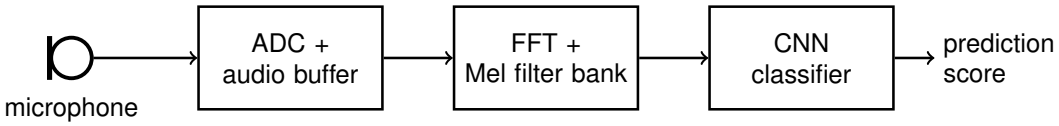


Figure 6.12 – Basic signal flow for real-time audio classification.

All audio signal processing is done in Python using the pyAudio Toolkit [115] and librosa [86]. The classification is done using Keras [116] with Tensorflow [117] as backend. Figure 6.12 shows the basic signal flow for real-time audio classification.

Figure 6.13 demonstrates the performance of a CNN trained with *'degraded'* audio data (see section 6.8 above). The top graph shows the audio waveform for 8 seconds of audio recorded with a far-field microphone array in a domestic environment. For around 3.5s, music is playing, which results in characteristic horizontal lines in the Mel-spectrogram (second plot). At around 5s, a male speaker starts talking, resulting in the characteristic succession of horizontal lines in the lower spectrum, pauses and transient wide-band noise bursts. In between (3.5s - 5s), a transient environmental noise event occurs.

The third and fourth plot show the predicted score and a smoothed score respectively. The smoothed score in the lower plot is obtained by averaging the last $N = 5$ predictions (moving average smoothing).

As can be seen, besides short segments at the transitions from music to environmental noise at around 3s and environmental noise to speech in at around 5s, the CNN accurately detects music, environmental noise and speech. This indicates that it might be interesting to include such transitions in the training data in order to further increase performance. Overall the detection of speech and music using CNNs was very reliable and responsive.

6.5. MiniDSP USBStreamer, *https://www.minidsp.com/products/usb-audio-interface/usbstreamer* - accessed January 2018

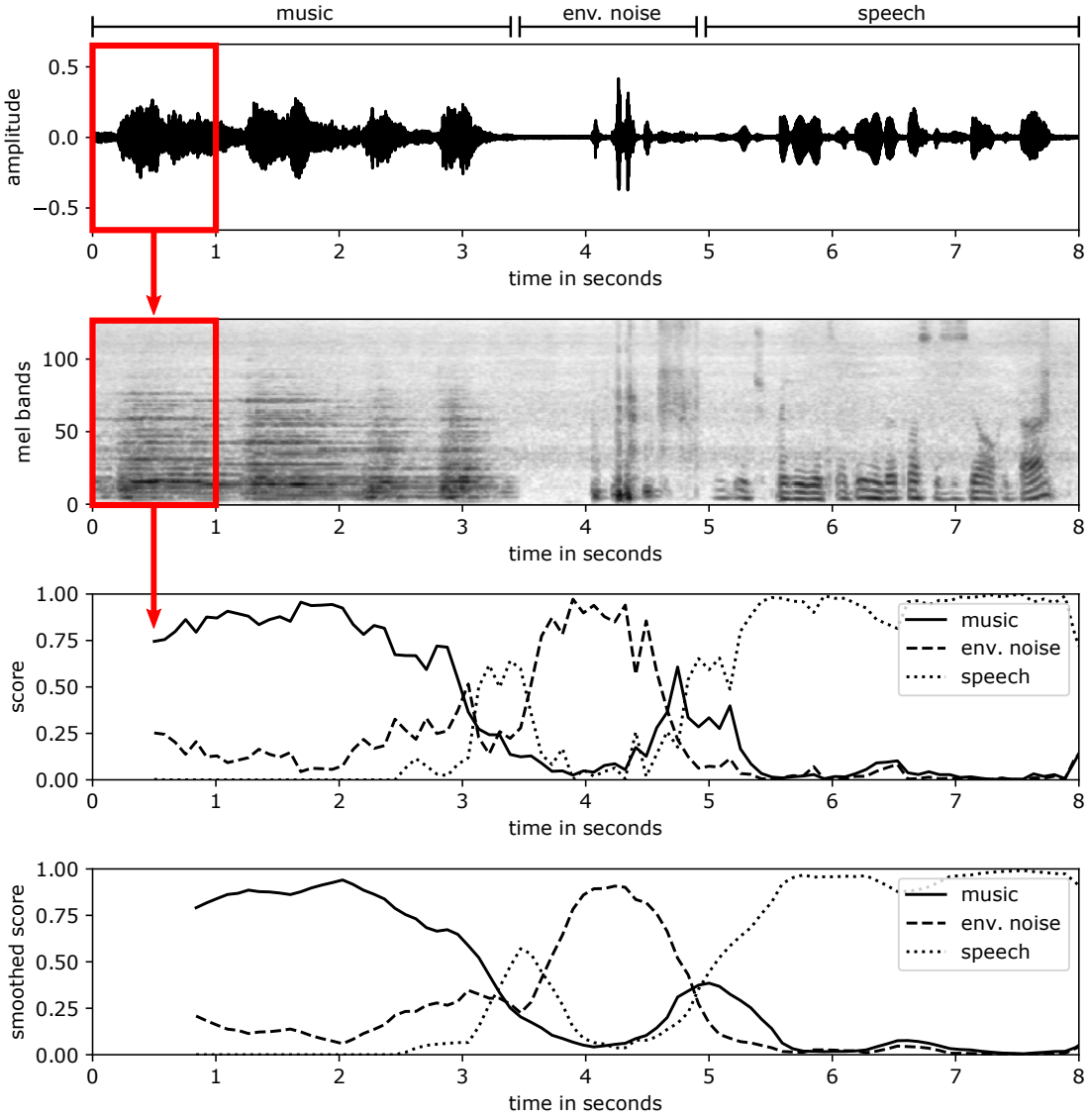


Figure 6.13 – Demonstration of classification performance on audio data stream. The top plot shows the audio waveform of a signal recorded with a far-field microphone array in a noisy environment. The class membership is estimated 12 times per second by evaluating the Mel-spectrogram (2nd plot) using a CNN. The predicted class membership score is shown in the third plot. The last plot shows the moving average of the predicted score, based on the last $N = 5$ predictions.

6.8.1 Directional Classification

When combining the audio classification algorithm with a beam-forming technique, different sources of audio surrounding the far-field microphone array can be detected in real-time. Figure 6.14 shows the basic signal flow of the real-time classification procedure integrating the beam-forming microphone array.

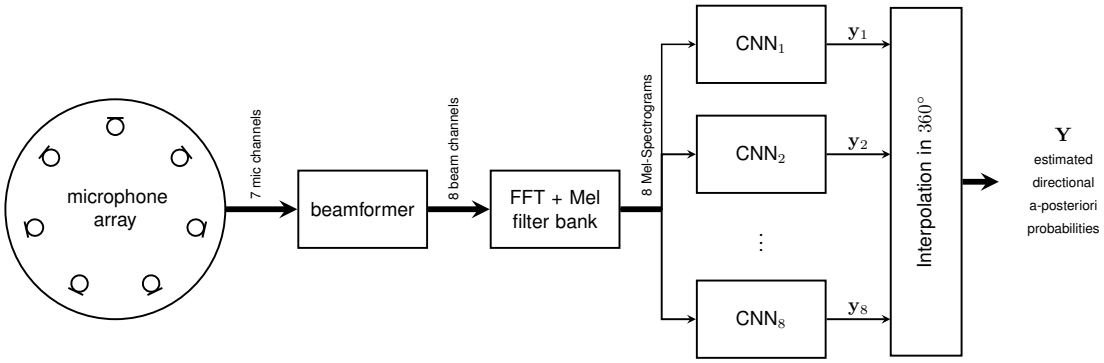


Figure 6.14 – microphone array with 7 omni-directional MEMS microphones.

In essence, 7 microphone audio channels are combined to 8 equally spaced beam channels using a filter-and-sum beam-forming technique. Mel-spectrograms are then computed for each beam channel. For each beam channel i, the corresponding a-posteriori probabilities $\mathbf{y}_i$ are predicted. Finally, the results for all 8 beams are interpolated to 360 degrees, yielding the output matrix $\mathbf{Y}$, containing the predicted a-posteriori probabilites $Y_{i,j} = \mathbb{P}(\alpha = i, c = j)$ for each direction $\alpha \in \{1, 2 \ldots, 360\}$ and class $c = \{0, 1, 2\}$ [6.6].

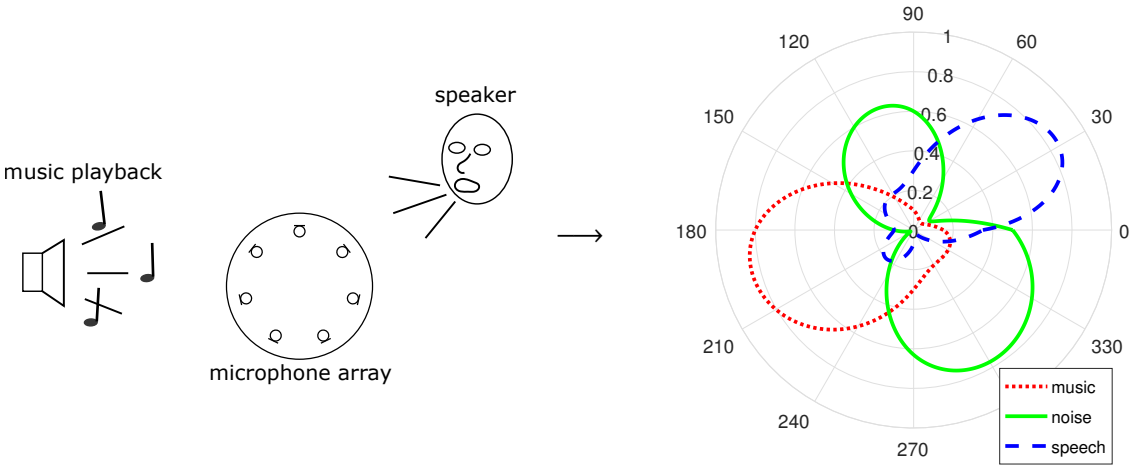


Figure 6.15 – Interpolated directional classification. The estimated a-posteriori probabilities for each class are interpolated to 360 degrees, yielding a predicted class membership for each direction.

6.6. Linear and cubic interpolation were evaluated, both with promising results.

In initial tests, up to three different sources could be separated to promising success, as qualitatively shown in figure 6.15. Separation of sources close to each other is limited by the directivity of beam channels. Nonetheless, the *'proof-of-concept'* implementation suggests that an audio classification algorithm could be useful for tracking various audio sources (like different speakers) in space, e.g. for adaptive beam steering.

7 | Summary

This thesis investigated the application of deep neural nets for audio classification with special emphasis on the performance in noisy and reverberant environments. More specifically, convolutional neural nets (CNNs) applied to Mel-spectrograms were used to discriminate music, speech and environmental noise in different domestic and business environments, outperforming other feature-based classifiers by a comfortable margin.

In this final chapter, experimental results are briefly summarised and an outlook is provided for future projects.

7.1 Summary of Experimental Results

Different audio classification algorithms were implemented and compared, including traditional feature-based machine learning (ML) algorithms, as well as state-of-the-art convolutional neural nets. Audio classification was done for audio segments of 1s. All algorithms were first trained and evaluated with *'clean'* low-noise audio data, resembling broadcast audio.

Both traditional ML algorithms and the investigated deep neural nets performed admirably for a *'clean'* test set (with accuracies of up to 98%). However, when evaluated with a *'target'* test set, resembling audio data recorded with a far-field microphone array in different noisy and reverberant locations, traditional feature-based approaches failed to discriminate speech and music from environmental noise effectively. CNNs on the other hand, performed quite well, even when trained with *'clean'* audio data only (with accuracies of up to 89%).

To overcome the lack of labelled audio data recorded with the investigated far-field microphone array, the *'clean'* training dataset was augmented by simulating the playback and recording in different acoustic environments (see chapter 4.5), forming the *'degraded'* training dataset. When trained with *'degraded'* audio data, performance for *'clean'* test data decreased slightly for all classifiers. More interestingly though, performance for the *'target'* test data improved significantly. However, performance for the feature-based classifiers was still poor (accuracy of up to 78%), especially compared to the impressive performance of the deep neural nets (accuracies of up to 94%).

Table 7.1 lists the classification accuracies for the different classifiers investigated for this thesis. As can be seen, the CNN outperforms all other classifiers investigated.

	train with *'clean'* data	
classifier:	test *'clean'*	test *'target'*
CNN	**98%**	**89%**
SVM	**98%**	67%
kNN	97%	58%
NB	90%	52%

	train with *'degraded'* data	
classifier:	test *'clean'*	test *'target'*
CNN	**95%**	**94%**
SVM	92%	77%
kNN	85%	65%
NB	68%	60%

Table 7.1 – Comparing different classifiers trained with *'clean'* and *'degraded'* train data and evaluated with both the *'clean'* and *'target'* test data. Table lists classification accuracy in percent.

To demonstrate the algorithm's capabilities, a prototype real-time system was implemented, that classifies an audio data stream coming from a far-field microphone array into the categories speech, music and environmental noise. Overlapping audio clips of 1s are evaluated multiple times per second. As suggested by the results of offline evaluations, the CNN classifier reliably detected speech and music, even when the source was several metres away from the microphone array. When combined with a beam-forming algorithm, audio data from various sources spaced around the microphone-array could be detected simultaneously with promising results.

7.2 Conclusion

As has been shown, deep neural nets are incredibly flexible and can be applied successfully to audio classification tasks. When trained with the right data, they provide very robust performance even when there is a considerable mismatch between training and evaluation data.

Deep neural nets learn to extract relevant features from raw input data (in our case Mel-spectrograms) during training. This means that less expert knowledge is required compared to traditional feature-based machine learning approaches, where descriptive audio features are defined based on a-priori domain knowledge.

Compared to simpler machine-learning algorithms, convolutional neural nets are computationally more demanding. So when computational resources and the amount of labelled training data are limited, simpler machine learning algorithms (like support vector machines) might yield a better trade-off between performance and computational effort.

Especially when acoustic environments are confined and well-known (such as for in-cabin applications in the automotive sector [26]), traditional feature-based

machine learning algorithms might yield good-enough performance.

However, due to the wide-spread application in computer vision (e.g. for autonomous driving [118]), a lot of specialised hardware for embedded training and evaluation of CNNs is actively developed and already available [119], indicating that computational cost will become less of a problem in the near future. With the current revival of analogue computing [120], deep neural nets might even find their way into hearing aids and other applications where space and power are limited [119, 121].

7.3 Outlook

In future work, the CNN model should be optimised for specific applications and embedded hardware implementations. Depending on the final use case and the availability of labelled training data, the model could be extended to discriminate more (sub-)categories.

This thesis evaluates only one specific type of deep learning algorithm, namely convolutional neural nets. While the results have been promising for applying this technology to new products in the near future (due to the wide-spread application in computer vision), other neural net topologies should be investigated as well. In particular so-called recurrent neural nets (RNNs) should be considered due to their ability to process time series [45, p.367].

The *'proof-of-concept'* real-time implementation of the audio classifier already reliably detects speech and music signals in reverberant and noisy environments. In future work, it should be investigated how the classification algorithm is affected by the use of de-noising, echo-cancelling and / or adaptive gain control algorithms. Also, using some sort of post-processing with regard to previous predictions might make the prediction more stable and reliable.

In general, deep neural nets require a lot of training data in order to find models that generalise well. More training data is thus always desirable. In future work, the training data should be augmented by labelled *'real-life'* audio data. Based on the performance gain achieved by using the *'degraded'* dataset instead of *'clean'* audio data for training, adding *'real-life'* data is expected to improve performance even further. Additionally, non-audio data could be included in the classification process as well. For example the location or time of day could provide valuable information.

The main focus of this thesis was on the evaluation of CNNs for audio classification. Other classification algorithms have been implemented to form a baseline to compare the CNN against, and were not investigated as thoroughly. In future work it might thus be interesting to see how a more extensive feature selection procedure might improve classification in noisy environments.

Bibliography

[1] E. Friauf, “Hearing,” *e-Neuroforum*, vol. 5, no. 3, pp. 51–52, 2014.

[2] A. S. Bregman, *Auditory Scene Analysis: The Perceptual Organization of Sound*. Cambridge, MA, USA: MIT Press, 1994.

[3] G. J. Brown and M. Cooke, “Computational auditory scene analysis,” *Computer Speech & Language*, vol. 8, no. 4, pp. 297–336, 1994.

[4] R. F. Lyon, “Machine hearing: An emerging field,” *IEEE signal processing magazine*, vol. 27, no. 5, pp. 131–139, 2010.

[5] “Introducing harman/kardon Invoke with Cortana by Microsoft,” Internet: http://www.harmankardon.com/invoke.html, accessed: February 2018.

[6] M. Kajala and M. Hamaldinen, “Broadband beamforming optimization for speech enhancement in noisy environments,” in *Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 1999, pp. 19–22.

[7] J. R. Clark, “Electronic device for automatically discriminating between speech and music forms,” U.S. Patent 2,761,897, Sept. 4 1956.

[8] J. R. Clark, “An automatic device for discriminating between speech and music,” *The Journal of the Acoustical Society of America*, vol. 23, pp. 148–149, 2006.

[9] J. Saunders, “Real-time discrimination of broadcast speech/music,” in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP-96, IEEE*, vol. 2, 1996, pp. 993–996.

[10] M. K. S. Khan and W. G. Al-Khatib, “Machine-learning based classification of speech and music,” *Multimedia Systems*, vol. 12, no. 1, pp. 55–67, 2006.

[11] E. Scheirer and M. Slaney, “Construction and evaluation of a robust multifeature speech/music discriminator,” in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP-97, IEEE*, vol. 2, 1997, pp. 1331–1334.

[12] Z. Liu, J. Huang, Y. Wang, and T. Chen, “Audio feature extraction and analysis for scene classification,” in *Proceedings of First Signal Processing Society Workshop on Multimedia Signal Processing*, 1997, pp. 343–348.

[13] T. Zhang and C. C.-J. Kuo, "Hierarchical classification of audio data for archiving and retrieving," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP-99, IEEE*, vol. 6, 1999, pp. 3001–3004.

[14] G. Williams and D. P. W. Ellis, "Speech/music discrimination based on posterior probability features," in *Proceedings of the Sixth European Conference on Speech Communication and Technology, Eurospeech '99*, 1999.

[15] K. El-Maleh, M. Klein, G. Petrucci, and P. Kabal, "Speech/music discrimination for multimedia applications," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP-00, IEEE*, vol. 4, 2000, pp. 2445–2448.

[16] M. De Santo, G. Percannella, C. Sansone, and M. Vento, "Classifying audio of movies by a multi-expert system," in *Proceedings of the 11th International Conference on Image Analysis and Processing, ICIAP-01, IEEE*, 2001, pp. 386–391.

[17] L. Lu, S. Z. Li, and H.-J. Zhang, "Content-based audio segmentation using support vector machines," in *Proceedings of the International Conference on Multimedia and Expo, ICME'01*, vol. 1, 2001, pp. 749–752.

[18] T. Zhang and C. C.-J. Kuo, "Audio content analysis for online audiovisual data segmentation and classification," *IEEE Transactions on speech and audio processing*, vol. 9, no. 4, pp. 441–457, 2001.

[19] M. Büchler, S. Allegro, S. Launer, and N. Dillier, "Sound classification in hearing aids inspired by auditory scene analysis," *EURASIP Journal on Advances in Signal Processing*, vol. 2005, no. 18, p. 387845, 2005.

[20] A. Bugatti, A. Flammini, and P. Migliorati, "Audio classification in speech and music: a comparison between a statistical and a neural approach," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 1, pp. 372–378, 2002.

[21] H. Harb and L. Chen, "Robust speech music discrimination using spectrum's first order statistics and neural networks," in *Proceedings of the Seventh International Symposium on Signal Processing and Its Applications, ISSPA-03, IEEE*, vol. 2, 2003, pp. 125–128.

[22] J. J. Burred and A. Lerch, "Hierarchical automatic audio signal classification," *Journal of the Audio Engineering Society*, vol. 52, no. 7/8, pp. 724–739, 2004.

[23] M. Khan, W. G. Al-Khatib, and M. Moinuddin, "Automatic classification of speech and music using neural networks," in *Proceedings of the 2nd ACM international workshop on Multimedia databases, MMDB'04*, 2004, pp. 94–99.

[24] L. Chen, S. Gunduz, and M. T. Ozsu, "Mixed type audio classification with support vector machine," in *Proceedings of the International Conference on Multimedia and Expo, ICME-06, IEEE*, 2006, pp. 781–784.

[25] Y. Lavner and D. Ruinskiy, "A decision-tree-based algorithm for speech/music classification and segmentation," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2009, no. 1, 2009.

[26] M. Won, H. Alsaadan, and Y. Eun, "Adaptive multi-class audio classification in noisy in-vehicle environment," *arXiv preprint arXiv:1703.07065*, 2017, [Online] Available: http://arxiv.org/abs/1703.07065, Accessed: January 2018.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Conference on Advances in Neural Information Processing Systems, NIPS'12*, 2012, pp. 1097–1105.

[28] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.

[29] Y. Lukic, C. Vogt, O. Dürr, and T. Stadelmann, "Speaker identification and clustering using convolutional neural networks," in *Proceedings of the 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2016, pp. 1–6.

[30] N. Takahashi, M. Gygli, and L. Van Gool, "Aenet: Learning deep audio features for video analysis," *IEEE Transactions on Multimedia*, vol. 20, no. 3, pp. 513–524, 2018.

[31] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, ICASSP-17, IEEE*, 2017, pp. 776–780.

[32] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold *et al.*, "Cnn architectures for large-scale audio classification," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, ICASSP-17, IEEE*, 2017, pp. 131–135.

[33] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of the International Symposium on Circuits and Systems, ISCAS'10, IEEE*, 2010, pp. 253–256.

[34] Y. Qian, M. Bi, T. Tan, and K. Yu, "Very deep convolutional neural networks for noise robust speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 12, pp. 2263–2276, 2016.

[35] Oxford University Press, "Oxford Living Dictionaries - English (Online Edition)," Internet: https://en.oxforddictionaries.com, 2017, accessed: July 2017.

[36] P. Vary and R. Martin, *Digital Speech Transmission: Enhancement, Coding And Error Concealment*. Hoboken, NJ, USA: John Wiley & Sons, 2006.

[37] B. Milner, *Display and Analysis of Speech*. New York, NY, USA: Springer New York, 2008, pp. 449–481.

[38] T. Houtgast and H. J. Steeneken, “A review of the MTF concept in room acoustics and its use for estimating speech intelligibility in auditoria,” *The Journal of the Acoustical Society of America*, vol. 77, no. 3, pp. 1069–1077, 1985.

[39] R. J. Baken and R. F. Orlikoff, *Clinical measurement of speech and voice*. Boston, MA, USA: Cengage Learning, 2000.

[40] International Phonetic Association, *Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet*. Cambridge, England: Cambridge University Press, 1999.

[41] “Wikipedia: Notation (Musik),” Internet: https://de.wikipedia.org/wiki/Notation_(Musik), 2017, accessed: December 2017.

[42] “Wikipedia: Tempo (Musik),” Internet: https://de.wikipedia.org/wiki/Tempo_(Musik), 2017, accessed: December 2017.

[43] E. Sengpiel, “Klaviatur und Frequenzen, Notennamen und Piano-Tastatur, Frequenzen der gleichstufigen Stimmung,” Internet: http://www.sengpielaudio.com/Rechner-notennamen.htm, accessed: December 2017.

[44] K. Seyerlehner, T. Pohle, M. Schedl, and G. Widmer, “Automatic Music Detection in Television Productions,” in *Proceedings of the Int. Conference on Digital Audio Effects, DAFX-2007*, Bordeaux, France, 2007.

[45] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[46] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the fifth annual workshop on Computational learning theory, ACM*, 1992, pp. 144–152.

[47] “Scikit Learn Python Library - Underfitting vs. Overfitting,” Internet: http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html, accessed: November 2017.

[48] H. Zhang, “The Optimality of Naive Bayes,” in *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004*, no. 2, 2004.

[49] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge, England: Cambridge University Press, 2008.

[50] “Scikit Learn Documentation - 1.9.1. Gaussian Naive Bayes,” Internet: http://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes, accessed: October 2017.

[51] “Scikit Learn Documentation - 1.6.2. Nearest Neighbors Classification,” Internet: http://scikit-learn.org/stable/modules/neighbors.html#classification, accessed: October 2017.

[52] C. Yeh, “Support Vector Machines for classification - Online Tutorial - efavdb.com (Everybody’s Favorite Data Blog),” Internet: http://efavdb.com/svm-classification/, 2015, accessed: October 2017.

[53] "Stackexchange.com: Data science - kernel trick explanation," Internet: https://datascience.stackexchange.com/questions/17536/kernel-trick-explanation, accessed: October 2017.

[54] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[55] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin, "Training and testing low-degree polynomial data mappings via linear svm," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1471–1490, 2010.

[56] "Pixabay.com - neuron," Internet: https://pixabay.com/de/neuron-nervenzelle-axon-dendrit-296581/, accessed: December 2017.

[57] P. H. Winston, "MIT Open Course Ware - Artificial Intelligence, lecture 12A: Neural Nets," Internet: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-12a-neural-nets/, 2010, accessed: July 2017.

[58] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the International Conference on Machine Learning, ICML-13*, vol. 30, no. 1, 2013, p. 3.

[59] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean *et al.*, "On rectified linear units for speech processing," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, ICASSP-13, IEEE*, 2013, pp. 3517–3521.

[60] C. Bishop, *Neural Networks for Pattern Recognition*, ser. Advanced Texts in Econometrics. Oxford, England: At the Clarendon Press, 1995.

[61] "MNIST For ML Beginners - tensorflow tutorial," Internet: https://www.tensorflow.org/versions/master/get_started/mnist/beginners, 2017, accessed: November 2017.

[62] "Wikipedia: Cross entopy," Internet: https://en.wikipedia.org/wiki/Cross_entropy, 2017, accessed: November 2017.

[63] C. Olah, "Visual Information Theory - Cross-Entropy," Internet: http://colah.github.io/posts/2015-09-Visual-Information/, 2015, accessed: November 2017.

[64] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[65] M. Nielsen, "Neural Nets and Deep Learning - Chapter 2: How the backpropagation algorithm works," Internet: http://neuralnetworksanddeeplearning.com/chap2.html, accessed: December 2017.

[66] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization," *presented at the International Conference on Learning Representations, ICLR, San Diego, CA*, 2015.

[67] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[68] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016, [Online] Available: http://arxiv.org/abs/1609.04747, Accessed: January 2018.

[69] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML'09*, 2009.

[70] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[71] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[72] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning, ICML-15*, 2015, pp. 448–456.

[73] "Scikit Learn Documentation - 3.3.2.4 Confusion Matrix," Internet: http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix, accessed: October 2017.

[74] "MIREX Grand Challenge 2016 web site," Internet: http://www.music-ir.org/mirex/gc16ux/index.php, 2016, accessed: June 2017.

[75] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, ICASSP-15, IEEE*, 2015, pp. 5206–5210.

[76] D. B. Dean, S. Sridharan, R. J. Vogt, and M. W. Mason, "The QUT-NOISE-TIMIT corpus for the evaluation of voice activity detection algorithms," in *Conference Proceedings of Interspeech*, 2010, pp. 3110–3113.

[77] D. Stowell, D. Giannoulis, E. Benetos, M. Lagrange, and M. D. Plumbley, "Detection and classification of acoustic scenes and events," *IEEE Transactions on Multimedia*, vol. 17, no. 10, pp. 1733–1746, Oct 2015.

[78] J. A. Stork, L. Spinello, J. Silva, and K. O. Arras, "Audio-based human activity recognition using non-markovian ensemble voting," in *Proceedings of the 21st International Symposium on Robot and Human Interactive Communication, RO-MAN2012, IEEE*, Sept 2012, pp. 509–514.

[79] A. Mesaros, T. Heittola, and T. Virtanen, "TUT database for acoustic scene classification and sound event detection," in *Proceedings of the European Signal Processing Conference, EUSIPCO*, 2016, pp. 1128–1132.

[80] J. Thiemann, N. Ito, and E. Vincent, "DEMAND: a collection of multi-channel recordings of acoustic noise in diverse environments," Internet: http://parole.loria.fr/DEMAND/, 2013, accessed: July 2017.

[81] L. M. Heller, "Sound events database," Internet: http://www.auditorylab.org/, 2008, accessed: July 2017.

[82] J. Beltrán, E. Chávez, and J. Favela, "Scalable identification of mixed environmental sounds, recorded from heterogeneous sources," *Pattern Recognition Letters*, vol. 68, pp. 153–160, 2015.

[83] J. Vettel, "CNBC Stimulus Repository - Sound Databases - Real World Events," Internet: http://wiki.cnbc.cmu.edu/Real_World_Events, accessed: July 2017.

[84] L. Lu, H.-J. Zhang, and S. Z. Li, "Content-based audio classification and segmentation by using support vector machines," *Multimedia Systems*, vol. 8, pp. 482–492, 2003.

[85] S. Ntalampiras, I. Potamitis, and N. Fakotakis, "Exploiting temporal feature integration for generalized sound recognition," *Eurasip Journal on Advances in Signal Processing*, no. May, 2009.

[86] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, 2015, pp. 18–25.

[87] SPTK Working Group, "The speech signal processing toolkit (sptk)," Internet: http://sp-tk.sourceforge.net, updated December 2016, accessed: October 2017.

[88] G. Peeters, B. L. Giordano, P. Susini, N. Misdariis, and S. McAdams, "The Timbre Toolbox: Extracting audio descriptors from musical signals," *The Journal of the Acoustical Society of America*, vol. 130, no. 5, pp. 2902–2916, 2011.

[89] "pysptk - a python wrapper for speech signal processing toolkit (sptk)," Internet: https://github.com/r9y9/pysptk, 2015, accessed: September 2017.

[90] G. Peeters, "A large set of Audio Features for Sound Description (similarity and classification) in the CUIDADO project," *CUIDADO Project Report*, 2004, [Online] Available: http://recherche.ircam.fr/anasyn/peeters/ARTICLES/Peeters_2003_cuidadoaudiofeatures.pdf, Accessed: July 2017.

[91] U. Zoelzer, Ed., *DAFX: Digital Audio Effects*. New York, NY, USA: John Wiley & Sons, 2002.

[92] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, "A tutorial on onset detection in music signals," *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 5, pp. 1035–1047, Sept 2005.

[93] C. Harte and M. Sandler, "Automatic chord identification using a quantised chromagram," *presented at the 118th Audio Engineering Society Convention, Barcelona, Spain*, 2005.

[94] M. Müller, S. Ewert, and S. Kreuzer, "Making chroma features more robust to timbre changes," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, ICASSP-09, IEEE*, April 2009, pp. 1877–1880.

[95] D. Ellis, "LabRosa - Matlab Resources: Chroma Feature Analysis and Synthesis," Internet: https://labrosa.ee.columbia.edu/matlab/chroma-ansyn/, 2007, accessed: July 2017.

[96] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[97] M. Slaney, "Auditory toolbox version 2 - technical report," Internet: https://engineering.purdue.edu/~malcolm/interval/1998-010/, 1998, accessed: July 2017.

[98] B. P. Bogert, M. J. R. Healy, and J. W. Tukey, "The quefrency alanysis of time series for echoes: Cepstrum, pseudo autocovariance, cross-cepstrum and saphe cracking," in *Proceedings of the Symposium on Time Series Analysis*, 1963, pp. 209–243.

[99] A. V. Oppenheim and R. W. Schafer, "From frequency to quefrency: a history of the cepstrum," *IEEE Signal Processing Magazine*, vol. 21, no. 5, pp. 95–106, Sept 2004.

[100] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey *et al.*, "The HTK book," *Cambridge university engineering department*, 2002.

[101] A. Camacho and J. G. Harris, "A sawtooth waveform inspired pitch estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 124, no. 3, pp. 1638–1652, 2008.

[102] H. F. Pollard and E. V. Jansson, "A tristimulus method for the specification of musical timbre," *Acustica*, vol. 51, pp. 162–171, 1982.

[103] L. Vrysis, N. Tsipas, C. Dimoulas, and G. Papanikolaou, "Extending Temporal Feature Integration for Semantic Audio Analysis," *presented at the 142nd Audio Engineering Society Convention*, 2017.

[104] "Rapid Miner Studio Documentation - Weight by Gini Index," Internet: https://docs.rapidminer.com/latest/studio/operators/modeling/feature_weights/weight_by_gini_index.html, accessed: October 2017.

[105] "Rapid Miner Studio Documentation - Weight by Information Gain," Internet: https://docs.rapidminer.com/latest/studio/operators/modeling/feature_weights/weight_by_information_gain.html, accessed: October 2017.

[106] "Rapid Miner Studio Documentation - Weight by Relief," Internet: https://docs.rapidminer.com/latest/studio/operators/modeling/feature_weights/weight_by_relief.html, accessed: October 2017.

[107] "Scikit Learn - Machine Learning in Python," Internet: http://scikit-learn.org/, accessed: October 2017.

[108] S. Dieleman and B. Schrauwen, "End-to-end learning for music audio," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, ICASSP-14, IEEE*, 2014, pp. 6964–6968.

[109] "Wikipedia: Cochlea," Internet: https://en.wikipedia.org/wiki/Cochlea, 2017, accessed: July 2017.

[110] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[111] A. J. Thomas, M. Petridis, S. D. Walters, S. M. Gheytassi, and R. E. Morgan, "Two hidden layers are usually better than one," in *International Conference on Engineering Applications of Neural Networks*. Springer, 2017, pp. 279–290.

[112] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, AISTATS*, 2010, pp. 249–256.

[113] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013, [Online] Available: http://arxiv.org/abs/1312.6034, Accessed: January 2018.

[114] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5188–5196.

[115] H. Pham, "PyAudio - cross-platform audio i/o library," Internet: http://people.csail.mit.edu/hubert/pyaudio/, 2006, accessed: August 2017.

[116] "Keras: The Python Deep Learning Library," Internet: https://keras.io/, accessed: December 2017.

[117] "Tensorflow: An open-source machine learning framework for everyone," Internet: https://www.tensorflow.org/, accessed: December 2017.

[118] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[119] Mythic, "www.mythic-ai.com - The future of local AI," Internet: http://www.mythic-ai.com/technology/, accessed: December 2017.

[120] E. Yablonovitch, "Deep Learning; the Reincarnation of Analog Computing - Brain Storming EECS Colloquium," Internet: https://eecs.berkeley.edu/research/colloquium/171115, 2017, accessed: December 2017.

[121] T. Simonite, "An old technique could put artificial intelligence in your hearing aid," Internet: https://www.wired.com/story/an-old-technique-could-put-artificial-intelligence-in-your-hearing-aid/, 2017, accessed: December 2017.