Technische Universität Ilmenau
Fakultät für Elektrotechnik und Informationstechnik

TECHNISCHE UNIVERSITÄT ILMENAU

Fraunhofer IDMT

Application of Automatic Speech Recognition Technologies to Singing

Doctoral Thesis

Submitted by: Anna Marie Kruspe

Date & place of birth: July 6, 1987, Halle/Saale

Course of study: Media Technology

Matriculation Number: 39909

Advisor: Prof. Dr.-Ing. Dr. rer. nat. h.c. mult. Karlheinz Brandenburg

Abstract

The research field of Music Information Retrieval is concerned with the automatic analysis of musical characteristics. One aspect that has not received much attention so far is the automatic analysis of sung lyrics. On the other hand, the field of Automatic Speech Recognition has produced many methods for the automatic analysis of speech, but those have rarely been employed for singing so far. This thesis analyzes the feasibility of applying various speech recognition methods to singing, and suggests adaptations. In addition, the routes to practical applications for these systems are described. Five tasks are considered: Phoneme recognition, language identification, keyword spotting, lyrics-to-audio alignment, and retrieval of lyrics from sung queries. The main bottleneck in almost all of these tasks lies in the recognition of phonemes from sung audio. Conventional models trained on speech do not perform well when applied to singing. Training models on singing is difficult due to a lack of annotated data. This thesis offers two approaches for generating such data sets. For the first one, speech recordings are made more "song-like". In the second approach, textual lyrics are automatically aligned to an existing singing data set. In both cases, these new data sets are then used for training new acoustic models, offering significant improvements over models trained on speech.

Building on these improved acoustic models, speech recognition algorithms for the individual tasks were adapted to singing by either improving their robustness to the differing characteristics of singing, or by exploiting the specific features of singing performances. Examples of improving robustness include the use of keyword-filler HMMs for keyword spotting, an i-vector approach for language identification, and a method for alignment and lyrics retrieval that allows highly varying durations. Features of singing are utilized in an approach for language identification that is well-suited for long recordings, in a method for keyword spotting based on phoneme durations in singing, and in an algorithm for alignment and retrieval that exploits known phoneme confusions in singing.

Zusammenfassung

Das Gebiet des Music Information Retrieval befasst sich mit der automatischen Analyse von musikalischen Charakteristika. Ein Aspekt, der bisher kaum erforscht wurde, ist dabei der gesungene Text. Auf der anderen Seite werden in der automatischen Spracherkennung viele Methoden für die automatische Analyse von Sprache entwickelt, jedoch selten für Gesang. Die vorliegende Arbeit untersucht die Anwendung von Methoden aus der Spracherkennung auf Gesang und beschreibt mögliche Anpassungen. Zudem werden Wege zur praktischen Anwendung dieser Ansätze aufgezeigt. Fünf Themen werden dabei betrachtet: Phonemerkennung, Sprachenidentifikation, Schlagwortsuche, Text-zu-Gesangs-Alignment und Suche von Texten anhand von gesungenen Anfragen. Das größte Hindernis bei fast allen dieser Themen ist die Erkennung von Phonemen aus Gesangsaufnahmen. Herkömmliche, auf Sprache trainierte Modelle, bieten keine guten Ergebnisse für Gesang. Das Trainieren von Modellen auf Gesang ist schwierig, da kaum annotierte Daten verfügbar sind. Diese Arbeit zeigt zwei Ansätze auf, um solche Daten zu generieren. Für den ersten wurden Sprachaufnahmen künstlich gesangsähnlicher gemacht. Für den zweiten wurden Texte automatisch zu einem vorhandenen Gesangsdatensatz alignt. Die neuen Datensätze wurden zum Trainieren neuer Modelle genutzt, welche signifikante Verbesserungen gegenüber sprachbasierten Modellen bieten.
Auf diesen verbesserten akustischen Modellen aufbauend wurden Algorithmen aus der Spracherkennung für die verschiedenen Aufgaben angepasst, entweder durch das Verbessern der Robustheit gegenüber Gesangscharakteristika oder durch das Ausnutzen von hilfreichen Besonderheiten von Gesang. Beispiele für die verbesserte Robustheit sind der Einsatz von Keyword-Filler-HMMs für die Schlagwortsuche, ein i-Vector-Ansatz für die Sprachenidentifikation sowie eine Methode für das Alignment und die Textsuche, die stark schwankende Phonemdauern nicht bestraft. Besonderheiten von Gesang werden ausgenutzt in einem Ansatz für die Sprachenidentifikation, der lange Aufnahmen benötigt, in einer Methode für die Schlagwortsuche, die bekannte Phonemdauern in Gesang mit einbezieht, sowie in einem Algorithmus für das Alignment und die Textsuche, der bekannte Phonemkonfusionen verwertet.

Table of Contents

1 Introduction

1.1 Motivation

Ever since the widespread introduction of digital formats for music, professional and personal music collections have grown exponentially. Efficient search algorithms are necessary for managing these huge collections. In the past ~15 years, many interesting technologies have been developed to make it easier for users to efficiently search these collections by certain semantic criteria, such as tempo, mood, genre, instruments, etc. This field of research is called Music Information Retrieval (MIR) [1]. One characteristic that has not received much research attention yet is the lyrical content of songs, even though this information is useful for many practical applications, and could aid other MIR tasks.

On the other hand, Automatic Speech Recognition (ASR) has been an active field of research for more than 60 years now [2] and encompasses a large variety of research topics. However, speech recognition algorithms have so far only rarely been adapted to singing. One of the reasons for this seems to be that most of these tasks get harder when using singing because singing data has different characteristics, which are also often more varied than in pure speech [3]. For example, the typical fundamental frequency for female speech lies between 165 and 200Hz, while in singing it can reach more than 1000Hz. Other differences include harmonics, durations, pronunciation, and vibrato.

Generally, both fields of research are strongly related and utilize many of the same approaches and technologies. This overlap, however, has not been explored thoroughly. This work aims to look at this relation more closely, and to apply and adapt ASR technologies to singing.

The possibilities for practical use of such technologies are manifold. Such applications include, but are not limited to:

Direct search of songs based on their lyrical content Potential users could search for songs by their language, lyrical phrases, or keywords. This is useful for such applications as language learning, finding songs on certain topics, advertisement etc.

Improvement of similarity search and playlist generation Similarity dimensions could include the sung language, keywords, or topics.

Improvement of regional classification As described in [4], human subjects tend to rely on the language to determine the region of origin of a musical piece. This is not taken into account by current regional classification systems.

Improvement of genre classification Similar to regional classification, certain musical genres are closely connected to a single singing language, or to certain keywords. Considering the "glass ceiling" of approximately 80% accuracy for many classification tasks in MIR [5], new hybrid approaches are necessary to improve them.

Improvement of mood detection Words can be indicatory of specific moods. By exploiting those, mood detection in music could be expanded with an additional dimension.

Lyrics alignment for karaoke Automatically aligned lyrics could be used in karaoke systems to enable users to sing any song they want to.

Lyrics retrieval from databases Textual lyrics can be retrieved from a database with just a short sung recording as the input. This is, once again, useful for karaoke.

Lyrics identification It is possible to compute compressed representations of detected lyrics. These could be used to aid audio identification (query-by-humming) technologies by utilizing lyrics information in addition to the melodic and harmonic characteristics used so far.

Cover song detection by lyrics In the same vein, alternative or auxiliary technologies for cover song detection through lyrics analysis are possible.

Lyrics transcription Given an audio recording, it will eventually be possible to automatically transcribe the full lyrics for users.

Singing generation Similar to recent approaches for speech synthesis [6], ASR technologies for singing could be used to automatically generate singing audio.

1.2 Research objectives

As described, speech recognition becomes more difficult when applied to singing. This thesis highlights strategies to improve various recognition tasks. There are two main starting points for this improvement:

1. Training better acoustic models for phoneme recognition, which forms the basis of many speech recognition tasks
2. Adapting subsequent algorithms for the various tasks to singing by either making them more robust to singing characteristics, or by exploiting knowledge about sung performances

To show how these improvements can impact speech recognition in singing, five topics were researched for this thesis:

Phoneme recognition Phoneme recognition describes the task of determining the sung sounds (phonemes) occurring in a an audio recording. This forms the basis for many other tasks; first and foremost, lyrics transcription, but also almost all other tasks in this work. Phoneme recognition tends to be the bottleneck component in systems for ASR in singing. Inaccurate results at this step will lead to inaccurate results in the subsequent ones.
As will be shown, phoneme recognition in singing has so far been performed with models trained on speech; these are, of course, not optimal. The reason why models have so far not been trained on singing is the lack of available training data for this task. This thesis presents ways around this problem. Specifically, acoustic models are trained on speech data that has been made more "song-like", and on singing data with automatically generated phoneme annotations. These new models lead to improvements on this and all the following tasks.

Language identification Language identification is the task of detecting the language in which a sung recording is performed. This has many practical applications, as described above: The results could be employed to directly search for music in certain languages (e.g. for language learning or for advertisements), to improve similarity search algorithms, or to support regional and genre classification.
There are very few publications dealing with sung language identification so far. In this thesis, a state-of-the-art approach from the field of Automatic Speech Recognition (ASR) is applied to the problem, and a completely new one based on phoneme statistics is presented.

Keyword spotting During keyword spotting, a set of singing recordings is searched for a specific keyword. Just like language identification, there are practical motivations for this. Keyword-based search systems are useful for finding songs on certain topics, for playlist generation, similarity search, genre classification, or for mood detection.
Once again, there are very few published approaches for this task. This thesis presents the first approach for English-language keyword spotting of arbitrary keywords without side information (like the musical score or sung samples of the keyword). Additionally, a new method for integrating knowledge about plausible phoneme durations is described.

Lyrics-to-audio alignment Using an audio recording and its known textual lyrics, lyrics-to-audio alignment methods are able to determine where each phrase, word, or phoneme occurs in time. In contrast to the other tasks, this topic is already relatively well-researched. It is a sought-after technology for karaoke applications, or for supporting other speech recognition tasks (for example, keyword spotting becomes much easier when the textual lyrics are available).
In this thesis, classic HMM-based alignment is first used as an auxiliary technology to create a new training data set for phoneme recognition. Then, two new methods are presented: One based on Dynamic Time Warping (DTW) on the results of the phoneme recognition, and one based on Levenshtein distance calculation on phoneme sequences.

Lyrics retrieval As another research topic that has not received much attention so far, lyrics retrieval is the task of finding the correct textual lyrics (and consequently the correct song) in a database given a sung query. This is, again, useful for karaoke systems or generally for voice-based search.
This work presents new approaches for this task that utilize the same technologies as the audio alignment. Compared to the state of the art, these are the first systems that do not require melody information in addition to the lyrics, and also work directly on the detected phonemes without a language modeling step required (which is, in effect, a text search on the detected phrases).

1.3 Thesis structure

The thesis is structured into nine chapters:

1 Introduction This chapter. Motivates the work and describes the research goals.

2 Technical background Describes the various algorithms for feature extraction, machine learning, and distance calculation used throughout this work. Also explains the general system structures of the developed approaches as well as their evaluation.

3 State of the art Summarizes existing methods for solving the mentioned research objectives.

4 Data sets An overview of the various data sets used for training and testing the developed approaches.

5 Singing phoneme recognition Describes the approaches developed for the phoneme recognition task and their evaluation results.

6 Sung language identification Presents the developed methods for language identification and their evaluation results.

7 Sung keyword spotting Explains the keyword spotting algorithms and their evaluation results.

8 Lyrics retrieval and alignment Describes the developed systems for lyrics alignment and retrieval and their evaluation results.

9 Conclusion Summarizes the work, points out the major contributions, and suggests future research directions.

1.4 Publications

The achieved research results have been published in the following conference papers:

Phoneme recognition

- Anna M. Kruspe, "Training phoneme models for singing with "songified" speech data", in *16th International Society for Music Information Retrieval Conference (ISMIR)*, Malaga, Spain, 2015.
- Anna M. Kruspe, "Bootstrapping a system for phoneme recognition and keyword spotting in unaccompanied singing", in *17th International Society for Music Information Retrieval Conference (ISMIR)*, New York, NY, USA, 2016.

Language identification

- Anna M. Kruspe, "Automatic Language Identification for Singing", in *Mid-Atlantic Student Colloquium on Speech, Language and Learning (MASC-SLL)*, Baltimore, MD, USA, 2013.
- Anna M. Kruspe, Jakob Abesser, Christian Dittmar, "A GMM approach to singing language identification", in *Proc. of the AES Conference on Semantic Audio*, London, UK, 2014.
- Anna M. Kruspe, "Improving singing language identification through i-vector extraction", in *Proc. of the 17th Int. Conference on Digital Audio Effects (DAFx-14)*, Erlangen, Germany, 2014.
- Anna M. Kruspe, "Phonotactic Language Identification for Singing", in *Interspeech*, San Francisco, CA, USA, 2016.

Keyword spotting

- Anna M. Kruspe, "Keyword spotting in a-capella singing", in *15th International Society for Music Information Retrieval Conference (ISMIR)*, Taipei, Taiwan, 2014.
- Anna M. Kruspe, "Keyword spotting in singing with duration-modeled HMMs", in *European Signal Processing Conference (EUSIPCO)*, Nice, France, 2015.
- Anna M. Kruspe, "Bootstrapping a system for phoneme recognition and keyword spotting in unaccompanied singing", in *17th International Society for Music Information Retrieval Conference (ISMIR)*, New York, NY, USA, 2016.

Lyrics alignment and retrieval

- Anna M. Kruspe, "Retrieval of textual song lyrics from sung inputs", in *Interspeech*, San Francisco, CA, USA, 2016.
- Anna M. Kruspe, "Automatic B**** Detection", in *17th International Society for Music Information Retrieval Conference (ISMIR)* (Late-breaking demo), New York, NY, USA, 2016.
- Anna M. Kruspe, Jakob Abesser, "Automatic lyrics alignment and retrieval from singing audio", in *Proc. of the AES Conference on Semantic Audio* (Late-breaking demo), Erlangen, Germany, 2017.

- Anna M. Kruspe, "Lyrics alignment using HMMs, posteriorgram-based DTW, and phoneme-based Levenshtein alignment", in *18th International Society for Music Information Retrieval Conference (ISMIR)* (*MIREX* submission), Suzhou, China, 2017.
- Anna M. Kruspe, M. Goto, "Retrieval of song lyrics from sung queries", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, Canada, 2018.

sehr schön, knapp aber gut verständlich

Frage: input polyphon <-> monophon ?

2 Technical Background

2.1 General processing chain

The general procedure for the tasks in this work is shown in figures 2.1 and 2.2. Using data sets of audio (speech or singing) and matching annotations, models are trained as outlined in figure 2.1. Then, these models are used to classify unseen audio data in order to generate annotations for them.
In detail, the necessary steps are:

Pre-processing For the tasks in this work, pre-processing of the audio data is relatively straightforward and consists of normalization of the signal and averaging to a mono channel. Additionally, the audio is usually downsampled to 16kHz because this is the lowest sampling frequency in most of the data sets, and downsampling is necessary for compatibility, except for the language identification data sets, which were downsampled to 8kHz (more detail on the data sets is given in chapter 4).

source separation ?

Feature extraction The audio signal contains a lot of data that is redundant and irrelevant to the tasks. For this reason, many types of so-called feature representations were developed over the years. The features employed in this work are described in the next section.

Model training Using both the audio features and the available annotations, models are trained with machine learning algorithms to gain an implicit understanding of the requested annotations (i.e. classes). In this work, only supervised learning was employed. The machine learning methods are described in section 2.4.

Classification The trained models can then be used on features extracted from unseen audio data to obtain their annotations (= classes).

Post-processing In many tasks, the classification results are not used directly, but processed further. In tasks like alignment and retrieval, for example, phoneme

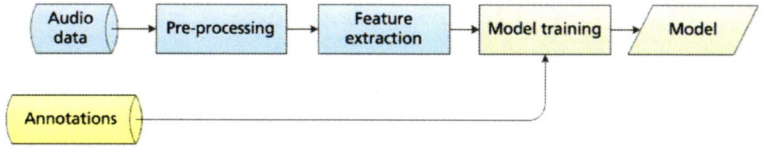


Figure 2.1: Schematic of the training procedure of the considered tasks.

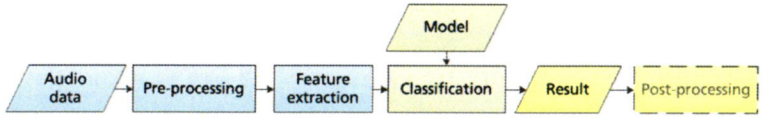


Figure 2.2: Schematic of the classification procedure of the considered tasks (the model is the one created during training - see figure 2.1).

probabilities are matched with symbolic phoneme annotations. In these cases, distance calculation methods as described in section 2.3 are required.

Implementation of these steps for the actual tasks is described in the individual chapters.

2.2 Audio features

This section describes the various audio features used througout this work. As one of the most successful features in ASR, Mel-Frequency Cepstral Coefficients (MFCCs) were used in all tasks, in some of them as the only feature. Shifted Delta Cepstrum (SDC) features, Perceptual Linear Prediction (PLP) features, and TempoRAl Patterns (TRAPs) were used in language identification. All features were extracted with a time resolution of 10ms, with window sizes of 20 to 25ms.

2.2.1 Mel-Frequency Cepstral Coefficients (MFCCs)

MFCCs are among the most frequently used audio features in speech recognition and Music Information Retrieval [7][8]. They were first introduced in 1976 by Mermelstein and Davis [9][10] based on previous experiments by Bridle and Brown [11].

The basic idea behind MFCCs comes from experiments in human auditory perception. Audio signals are transformed into a representation that is based on human perceptual sensitivities. This is done by taking the Short-Term Fourier Transform (STFT) of an audio signal and then mapping the resulting spectrum from a linear frequency scale onto a Mel scale. Then, the Discrete Cosine Transform (DCT) of the resulting log energies is calculated along the frequency axis to decorrelate the spectral band signals. The result is a representation of the various frequencies within the spectrum (i.e. the cepstrum), which can be interpreted as ranging from the spectral envelope to the more fine-grained components. In theory, this makes the result largely independent of the absolute frequencies (i.e. the pitch), but representative of the perceptual content (e.g. phonemes). One point of criticism, however, is the lack of interpretability of the MFC coefficients.

In detail, the calculation is performed as follows:

1. **Short-term Fourier transform (STFT)** After cutting a signal s into frames $s_i, i = 0, 1, ..., I$ (e.g. of 10ms duration) and windowing it (e.g. with a Hamming window), the Discrete Fourier Transform (DFT) is calculated for each time frame:

$$S_i(k) = \sum_{n=0}^{N-1} s_i(n)h(n)e^{-j2\pi kn/N}, k = 0, 1, ..., K-1 \tag{2.1}$$

where $h(n)$ is the window of length N, and K is the DFT length. For the further calculations, only the power spectrum is used:

$$P_i(k) = \frac{1}{N} \mid S_i(k) \mid^2 \tag{2.2}$$

2. **Mel-spectrum calculation** The resulting energies are mapped from the linear frequency scale to a perceptually motivated Mel scale. This is done by convolving the spectrum with a set of M triangular Mel-spaced filters $H_m(k)$, such as the ones shown in figure 2.3. Furthermore, the resulting energy outputs are logarithmized:

$$X_m = \log_{10}\left(\sum_{k=0}^{K-1} \mid P_i(k) \mid \cdot H_m(k)\right), m = 1, 2, ..., M \tag{2.3}$$

3. **Discrete Cosine Transform (DCT)** Finally, the Mel-scale spectrum is transformed

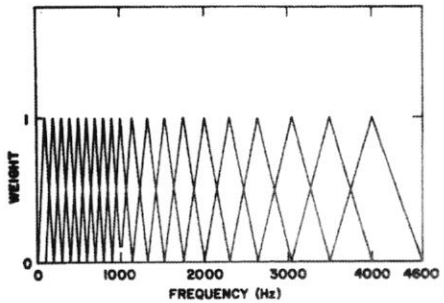


Figure 2.3: Example of a Mel filterbank. [10]

with a DCT, resulting in the so-called cepstrum:

$$C_j = \sum_{m=1}^{M} X_m \cdot \cos\left((j+1)\cdot(m-1/2)\cdot\frac{\pi}{M}\right), j = 0, 1, ..., J-1 \tag{2.4}$$

The J MFC coefficients are retained as features. The 0th coefficient can be interpreted as the power over all frequency bands, and the 1st coefficient as the global energy balance between low and high frequencies [12].

In this work, 13 coefficients including the 0th coefficient are extracted, with the exception of language identification, where 20 coefficients are used. In addition, deltas and double-deltas are calculated to capture information about the feature's trajectory:

$$\Delta(C(n)) = C(n) - C(n-1) \tag{2.5}$$

$$\Delta\Delta(C(n)) = \Delta(C(n)) - \Delta(C(n-1)) \tag{2.6}$$

2.2.2 Shifted Delta Cepstrum (SDCs)

Shifted Delta Cepstrum features were first described in [13] and have since been successfully used for speaker verification and language identification tasks on speech data [14] [15] [16]. They are calculated on MFCC vectors and take their temporal evolution into account. This has been shown to improve recognition results because speech and

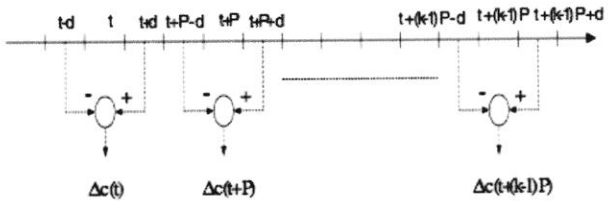


Figure 2.4: The SDC calculation at frame t. [14]

singing signals are defined by their temporal contexts. Their configuration is described by the four parameters $N-d-P-k$, where N is the number of cepstral coefficients for each frame, d is the time context (in frames) for the delta calculation, k is the number of delta blocks to use, and P is the shift between consecutive blocks. The delta cepstrals are then calculated as:

$$\Delta c(t) = C(t+iP+d) + C(t+iP-d), 0 \leq i \leq k \tag{2.7}$$

with $C = (C_0, C_1, ..., C_{N-1})$ as the previously extracted cepstral coefficients. The resulting k delta cepstrals for each frame are concatenated to form a single SDC vector of the length kN. In this work, the common parameter combination $N = 7, d = 1, P = 3, k = 7$ was used. The calculation is visualized in figure 2.4.

2.2.3 Perceptual Linear Predictive features (PLPs)

PLP features, first introduced in [17], are also among the most frequently used features in speech processing, next to MFCCs. They are based on the idea to use knowledge about human perception to emphasize important speech information in spectra while minimizing the differences between speakers.

In principle, these ideas are related to those that MFCCs are based on, but knowledge about human perception is integrated more extensively. A comparison of the steps of both algorithms is given in figure 2.5. For PLP computation, these steps are as follows:

1. **Short-term Fourier transform** As in MFCC extraction, the signal s is segmented into frames, which are then windowed, and the STFT is calculated for each of them. The power spectrum is X is used for further processing.

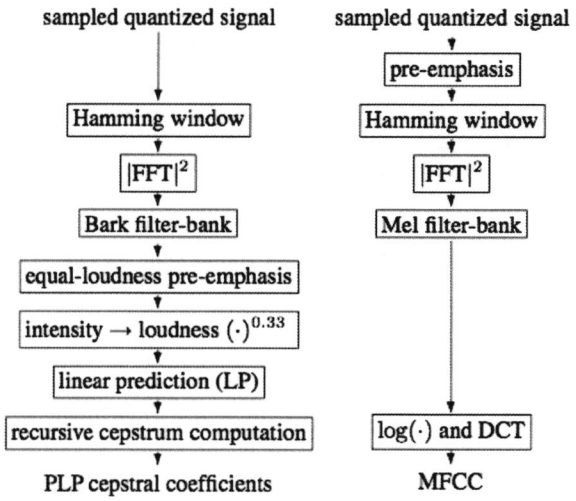


Figure 2.5: Comparison of the processing steps in PLP (left) and MFCC (right) calculation. [18]

2. **Bark-spectrum calculation** Similar to the Mel-frequency transformation step, the resulting energies P are mapped to a Bark frequency scale, which is also perceptually motivated (resulting in Bark-scaled energies $X_i(\omega)$). As described in [19] and [18], there is no necessity for the particular use of a Bark scale, but it is employed for historic reasons. A comparison of the filters of which Mel and Bark filterbanks are composed is shown in figure 2.6. Furthermore, the coefficients are logarithmized.

3. **Equal loudness pre-emphasis** The filterbank coefficients are weighted with an equal-loudness curve $E(\omega)$ which simulates the varying sensitivities of human hearing across the frequency range. Figure 2.7 displays such a curve; Makhoul and Cosell presented a numerical approximation [20]. (As mentioned in figure 2.5, such a pre-emphasis is sometimes performed as the first step of MFCC calculation as well). This is computed as

$$\Xi(\omega) = X_i(\omega) \cdot E(\omega) \tag{2.8}$$

4. **Intensity - loudness conversion** This steps integrates knowledge about the relationship between the intensity of the signal and its perceived loudness. According to the power law of hearing [21], this relation can be approximated as a cubic root compression:

$$\Phi(\omega) = \Xi(\omega)^{0.33} \tag{2.9}$$

5. **Autoregressive modeling** An inverse DFT is then applied to the computed loudness signal to obtain an auto-correlation function. Then, the actual linear prediction is implemented with an all-pole model as described in [22]. Levinson-Durbin recursion is employed to compute the final PLP coefficients from the auto-correlation function.

Later, RASTA filtering was introduced as a step between the Bark-spectrum calculation and the equal loudness pre-emphasis (i.e. steps 2 and 3) [23]. This is essentially a bandpass filtering in the log-spectral domain that serves to suppress the slow-changing components of the signal, which are commonly rooted in the transmission channel rather than the content. The filter is defined as

$$H(z) = 0.1\frac{2 + z^{-1} - z^{-3} - 2z^{-4}}{z^{-4} \cdot (1 - 0.98z^{-1})} \tag{2.10}$$

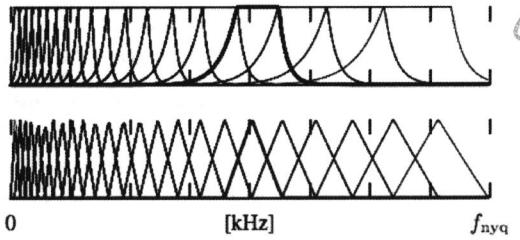


Figure 2.6: Mel-scale (top) and Bark-scale filterbank. [18]

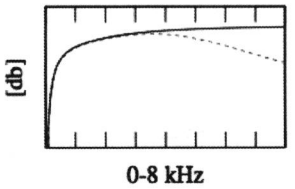


Figure 2.7: Perceptually motivated equal-loudness weighting function. (The dashed function is used for signals with a Nyquist frequency >5kHz). [18]

In this work, model orders of 13 and 36 are used. Deltas and double-deltas between frames are also calculated, and PLPs are tested with and without RASTA preprocessing.

2.2.4 TempoRal Patterns (TRAP)

TRAPs were developed by Hermansky [24] [25] and have also been used successfully in a number of speech recognition tasks. In contrast to MFCCs and PLPs, which, apart from delta calculation, only consider a single spectral frame at a time, TRAPs take the spectral development over time into account. This is visualized in figure 2.8. To demonstrate the feature's suitability for phoneme classification, examples of mean TRAPs for various phonemes in the 5th critical band are shown in figure 2.9.

In [26], a slightly modified method is presented. An overview of the steps necessary for this version of TRAP calculation is given in figure 2.10. In detail, these are:

1. **Grouping into spectral bands** Spectral band signals are extracted from the signal with a triangular Mel-scale filterbank, such as the one presented in figure 2.3. The log-energy of each band is used for further processing.

2. **Normalization and windowing** Each band's trajectory is normalized and windowed with relatively long windows (e.g. Hamming windows with a temporal context of 200 to 1000ms) to obtain a representation of the temporal development.

3. **DCT decorrelation** A DCT is applied to each frame to decorrelate its coefficients and reduce dimensionality. The vectors for each critical band are concatenated. (In classic TRAP calculation, separate classifiers would be trained for each band in this step).

4. **Model training** In classic TRAP calculation, the resulting band coefficients are now used to train a Multilayer Perceptron with a single hidden layer to obtain phoneme probabilities [26]. However, as suggested in [27], the feature values extracted so far can also be used to train other models, or even be combined with other features beforehand. In [28], the authors suggest that a combination with MFCCs works particularly well as these two features cover different sets of characteristics: MFCCs are better at capturing the spectral content, while TRAPs model the temporal progressions better.

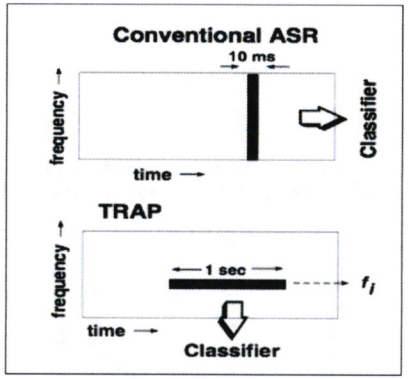


Figure 2.8: The temporal paradigm for TRAP extraction versus conventional features (e.g. MFCC). [24]

In this work, the coefficient vector is used directly as a feature to train various models. 8 linear spectral bands were extracted with a time context of 20 frames (corresponding to 200ms), and the first 8 DCT coefficients were kept.

2.3 Distance calculation

In this section, two algorithms for distance calculation used in this work are described. Dynamic Time Warping (DTW) is used for calculating optimal alignments between two sequences of continuous values, while Levenshtein alignment is particularly useful for finding the optimal alignment (and therefore the minimum distance) between two sequences with discrete values, such as character strings, when allowing deletions, insertions, and replacements.

2.3.1 Dynamic Time Warping

Dynamic Time Warping (DTW) is an algorithm for finding an optimal alignment between two time sequences of vectors X and Y, which was originally developed for aligning speech sequences to each other [29]. X and Y are not required to have the same length (i.e. $X = (x_1, ..., x_M)$ and $Y = (y_1, ..., y_N)$). To this end, varying durations of parts of each sequence are allowed. The result is a warping path $W = (w_1, ..., w_K)$ where each element represents an alignment of the elements of the two sequences (i.e.

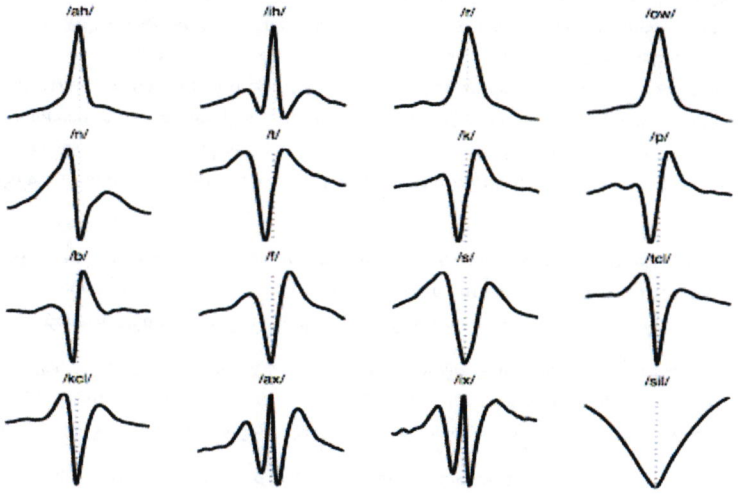


Figure 2.9: Mean TRAPs of various phonemes in the 5th critical band. [24]

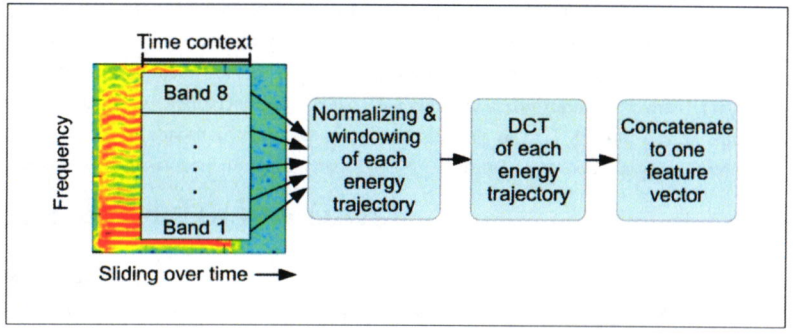


Figure 2.10: TRAP extraction process. [27]

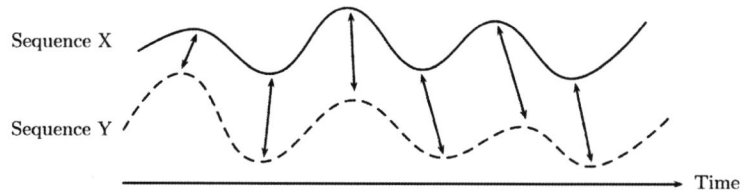


Figure 2.11: Example of a DTW alignment. Alignment between points is represented by the arrows. [8]

$w_k = (m_k, n_k)$ represents the alignment of the elements x_{m_k} and y_{n_k} of X and Y).
In classic DTW, the warping path must fulfill three restrictions:

1. **Boundary condition** The warping path must align the whole sequences to each other - i.e. $w_1 = (1,1)$ and $w_K = (M,N)$.

2. **Step size condition** The warping path may only step sequentially forward in either direction - i.e. $w_{k+1} - w_k \in \{(0,1),(1,0),(1,1)\}$.

3. **Monotonicity condition** The warping path cannot skip backwards - i.e. $m_1 \leq m_2 \leq ... \leq m_K$ and $n_1 \leq n_2 \leq ... \leq n_K$. (This is, in fact, already implied by condition 2).

A graphic example of such an alignment is given in figure 2.11.

A DTW consists of two steps: Cost calculation and path detection. In the cost calculation steps, a local cost c is calculated for all pairs (x_m, y_n), resulting in a cost matrix C. An example is shown in figure 2.12. Common cost functions include the Manhattan distance and the cosine distance (i.e. the complement of the normalized inner product), which was used in this work:

$$c(x_m, y_n) = 1 - cos(\theta) \tag{2.11}$$

where θ is the angle between x_m and y_n.

In the second step, an optimal warping path is calculated on the cost matrix. The

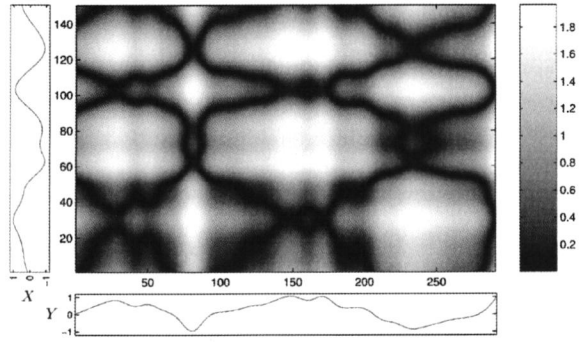


Figure 2.12: Example of a matrix of costs between two sequences X and Y using the Manhattan distance as the local cost measure. [8]

cost of a warping path is

$$c_W(X,Y) = \sum_{k=1}^{K} c(x_{m_k}, y_{n_k}) \tag{2.12}$$

and the optimal warping path is the one with minimal cost - i.e. the DTW cost:

$$DTW(X,Y) = \min\{c_W(X,Y) | W \text{ is a warping path}\} \tag{2.13}$$

Consequently, the DTW cost can also be used to compare the quality of alignments of one query sequence to multiple other sequences when taking the varying lengths into account. The path calculation is commonly solved using a Dynamic Programming algorithm [8]. In this work, the implementation from [30] is used.

Subsequence DTW is a variant of this algorithm in which the boundary condition is loosened. This means that the optimal warping path can run along a subsequence of the longer compared sequence instead of the full series. This is more computationally expensive since more paths need to be calculated for comparison.

2.3.2 Levenshtein distance

The Levenshtein distance, also called edit distance, is a measure of similarity between two strings (or character sequences). The algorithm was first described by Levenshtein in 1965 [31] and can also be used to retrieve an optimal alignment (so-called approximate string matching [32]). In that sense, it serves a similar purpose as DTW for strings instead of time sequences. This measure is commonly used in the fields of Computational Biology, Natural Language Processing, and signal processing.

The distance is the sum of character operations necessary to transform one string into the other. These operations can be substitutions, insertions, or deletions, which may be weighted differently. An example is shown in figure 2.13. If each operation has a cost of 1, the Levenshtein distance in this example is 5; if substitutions are weighted with a cost of 2, the distance is 8.

Just like DTW, this problem is usually solved efficiently with a Dynamic Programming approach. For two strings $X = (x_1, x_2, ..., x_M)$ and $Y = (y_1, y_2, ..., y_N)$, the initial step is then defined as

$$L(0,0) = 0, L(i,0) = \sum_{k=1}^{i} I(x_k), L(0,j) = \sum_{k=1}^{j} D(y_k) \tag{2.14}$$

and the recursive step is defined as

$$L(i,j) = \min \begin{cases} L(i-1,j) + D(y_j) \\ L(i,j-1) + I(x_i) \\ L(i-1,j-1) + S(x_i, y_j) \end{cases} \tag{2.15}$$

where $L(i,j)$ is the Levenshtein distance at step (i,j) and D, I, and S are the costs for deletions, insertions, and substitutions respectively [33]. The over-all Levenshtein distance is $L(M,N)$.

The Levenshtein distance is often employed to measure the quality of speech recognition systems with regards to the recognized phonemes or words. The so-called Phoneme Error Rate (PER) is simply the Levenshtein distance between a generated and an expected phoneme sequence where insertions, deletions, and replacements are weighted


```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Figure 2.13: Example of a Levenshtein distance calculation between the two strings "INTENTION" and "EXECUTION". Found operations (deletions, insertions, substitutions) are shown at the bottom. [34]

equally, normalized by the length of the expected sequence:

$$PER = \frac{D + I + S}{N} \tag{2.16}$$

where D are deletions, I are insertions, and S are substitutions of phonemes and N is the length of the sequence. (The accuracy measure used in some of the state-of-the-art works is the same as $1 - PER$; a Word Error Rate can be calculated analogously). Some works also use a measure called *correct*, which ignores insertions. This makes sense if it is assumed that the phoneme results are used afterwards by an algorithm that is tolerant to insertions. In many cases, such post-processing steps will then also be tolerant to deletions. For cases like this, Hunt suggested a weighted error rate that punishes insertions and deletions less heavily than substitutions [35]:

$$Weighted\ PER = \frac{0.5D + 0.5I + S}{N} \tag{2.17}$$

2.4 Machine learning algorithms

This section describes the various Machine Learning algorithms employed throughout this thesis. Gaussian Mixture Models (GMMs), Hidden Markov Models (HMMs), and Support Vector Machines (SVMs) are three traditional approaches that are used as the basis of many new approaches, and were used for several starting experiments. i-Vector processing is a relatively new, more sophisticated approach that bundles several other machine learning techniques.

In recent years, Deep Learning has become the standard for machine learning applications [36]. This chapter also describes a new approach that was used extensively in this work: Deep Neural Networks (DNNs).

2.4.1 Gaussian Mixture Models

In their basic form, Gaussian Mixture Models (GMMs) are a form of unsupervised learning. Given a set of observations $X = (x_1, x_2, ..., x_N)$, their probability distribution is modeled with a superposition of Gaussian distributions:

$$p(x_n|\lambda) = \sum_{i=1}^{M} p_i b_i(x) \tag{2.18}$$

where b_i are the constituting distributions, p_i are the mixture weights, and λ are the model parameters. A visualization is shown in figure 2.14. If the observations are multidimensional (which is the case for audio features), multivariate Gaussians (with D dimensions) are used for this:

$$b_i(x) = \frac{1}{(2\pi)^{D/2}\det(\Sigma_i)^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu_i)\Sigma_i^{-1}(x-\mu_i)\right\} \tag{2.19}$$

where μ_i is the mean vector and Σ_i is the covariance matrix.
These parameters, together with the mixture weights, define the model:

$$\lambda = \{p_i, \mu_i, \Sigma_i\}, i = 1, ..., M \tag{2.20}$$

For each observation o_j, the contribution of each Gaussian b_i can be calculated as:

$$p_{ni} = P(i|n) = \frac{b_i P(i)}{P(x_n)} \tag{2.21}$$

The overall likelihood of the model is:

$$L = \prod_{n=1}^{N} P(x_n) \tag{2.22}$$

In order to find the optimal parameters λ, the iterative Expectation Maximization (EM) algorithm is commonly used [37]. In the expectation step, L is calculated; in the maximization step, the parameters are adapted. This is repeated until convergence.

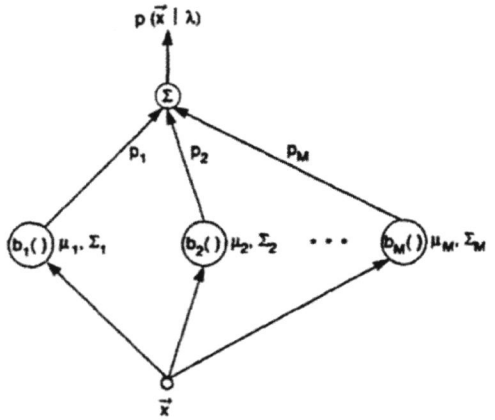


Figure 2.14: Visualization of a GMM. The Gaussian mixture is the weighted sum of several Gaussian distributions, where p_i are the mixture weights and b_i are the Gaussians. [38]

An example of such a training procedure is visualized in figure 2.15.

Gaussian Mixture Models have been used in many areas of machine learning, for example in ASR [38], in image retrieval [40], in financial modeling [41], and in visual tracking [42]. When used for classification, one GMM is trained for each class $S = (s_1, s_2, ..., s_J)$ separately, resulting in J sets of parameters λ. The likelihood L_j of each class is then determined, and the most likely class is chosen:

$$C = \underset{1\leq j\leq J}{\operatorname{argmax}} P(\lambda_j|X) = \underset{1\leq j\leq J}{\operatorname{argmax}} \frac{p(X|\lambda_j)P(\lambda_j)}{p(X)} \tag{2.23}$$

(according to Bayes' rule). If all classes and all observations are equally likely, this simplifies to

$$C = \underset{1\leq j\leq J}{\operatorname{argmax}}\, p(X|\lambda_j) \tag{2.24}$$

In practice, log-probabilities are commonly used for numerical reasons, resulting in the calculation:

$$C = \underset{1\leq j\leq J}{\operatorname{argmax}} \sum_{n=1}^{N} p(x_n|\lambda_j) \tag{2.25}$$

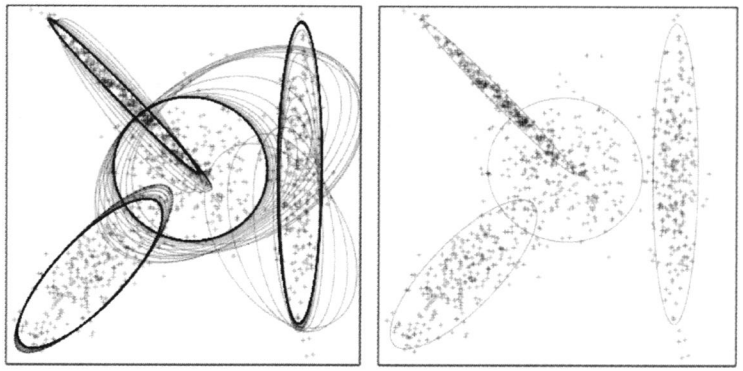


Figure 2.15: Example of a GMM in two dimensions. Crosses signify the data points, ellipses the three multivariate Gaussians. On the left-hand side, the evolution of the estimated means and covariances of these Gaussians during the EM process is shown; the right-hand side shows the converged mixture. [39]

In addition to this direct use for classification, GMMs are often used to model the emission probabilities in Hidden Markov Models; these are then called GMM-HMMs.

2.4.2 Hidden Markov Models

Markov models are statistical models of Markov processes - i.e. sequences of states in which the probability of each state only depends on the previous one. In a Hidden Markov Model (HMM), these states are not directly observable, but may be inferred from the models emissions. HMMs were first suggested by Baum et al. around 1970 [43][44][45][46][47]. Due to their ability to model temporal processes, they have been employed extensively in ASR [48][49][50][51]. Apart from this field, they are also frequently used in Natural Language Processing [52], Optical Character Recognition [53], and Computational Biology [54][55].

A HMM consists of four basic components:

- The observation (= emission) sequence $Y = (y_1, y_2, ..., y_T)$
- The hidden state nodes $Q = (q_1, q_2, ..., q_N)$; the sequence of hidden states corresponding to the observation sequence will be denoted as $I = (i_1, i_2, ..., i_T)$ where $i_t \in Q$.
- The transition probabilities between the hidden states, defined by a transition matrix $A \in \mathbb{R}^{N \times N}$; additionally, the initial state distribution (i.e. the probability of starting in each state) $\pi \in \mathbb{R}^N$
- The emission probabilities $B = (b_1(k), b_2(k), ..., b_N(k))$, mapping from the hidden states to the observations. These can, for example, be Gaussians for continuous outputs y_t, or conditional probabilities for discrete y_t.

The transition probabilities, initial state distribution, and emission probabilities define the model λ.

In the case of speech recognition, the observations are the feature vectors, and the hidden states are the phonemes generating these features. Such a model is visualized in figure 2.16. Different variants of HMMs can be created by restricting the transition matrix in certain ways; e.g., left-to-right HMMs, which only allow transitions to subsequent states and are often used in speech recognition and handwriting recognition [56]. A particularly interesting property of HMMs for speech recognition is their relative invariance to warping along the time axis because states can usually be repeated for arbitrary amounts of time.

Three problems commonly need to be solved for problems modeled with HMMs:

Evaluation - i.e., how probable is an observation sequence given this model? In mathematical terms, the probability $P(Y, I|\lambda)$ is sought. The most straightforward way to do this would be to calculate this probability for each possible Y of the length T of the observation sequence, but this is very computationally expensive. For this reason, an algorithm called *forward procedure* is used. A forward variable α representing the probability at time t is introduced:

$$\alpha_t(i) = P(y_1, y_2, ..., y_t, i_t = q_i|\lambda) \tag{2.26}$$

This can be solved inductively with the initialization

$$\alpha_1(i) = \pi_i b_i(Y_1), 1 \leq i \leq N \tag{2.27}$$

and the induction step

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_i j\right] b_j(Y_{t+1}) \tag{2.28}$$

This is a form of Dynamic Programming.

Training - i.e., how can the parameters λ be set optimally to maximize the probability of observed sequences? There is no analytical way to compute this, but the so-called *Baum-Welch* algorithm (which is a special case of the Expectation Maximization algorithm) allows for an iterative estimation of the parameters. In order to do this, a backward variable β is calculated analogous to α to represent the probability of the sequence from time $t+1$ to the end:

$$\beta_t(i) = P(y_{t+1}, y_{t+2}, ..., y_T, i_t = q_i|\lambda) \tag{2.29}$$

$$\beta_T(i) = 1, 1 \leq i \leq N \tag{2.30}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_i j b_j(Y_{t+1}) \beta_{t+1}(j) \tag{2.31}$$

The probability of a path being in state q_i at t and making a transition to q_j at $t+1$ is then:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_i j b_j(Y_{t+1} \beta_{t+1}(j)}{P(Y|\lambda)} \tag{2.32}$$

This can be used to calculate the expected numbers of transitions and emissions, which can be re-adapted with statistics from the observation sequences and used to adjust α and β. This process is repeated until convergence.

Decoding - i.e., given an observation sequence, what is the most probable underlying sequence of hidden states? This is particularly interesting for speech recognition since the interpretation here is the detection of the phonemes generating a series of feature vectors.

Again, this problem is broken down by first defining a variable for the probability of being in state q_i at time t:

$$\gamma_t(i) = P(i_t = q_i|Y, \lambda) \tag{2.33}$$

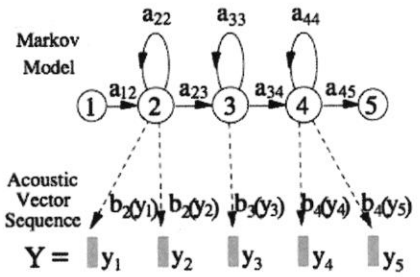


Figure 2.16: A HMM as it is commonly used in ASR, with phonemes as hidden states and acoustic feature vectors as the observations. $a_12, a_22, a_23, ...$ are elements of the transition matrix A; $b_2(y1), b_2(y_2), b_3(y_3)$ are elements of the output probability matrix B; $Y = y_1, y_2, ...$ is the observation sequence. [51]

and therefore, the most likely state at t is:

$$i_t = \underset{1 \leq i \leq N}{\operatorname{argmax}}[\gamma_t(i)], 1 \leq t \leq T \tag{2.34}$$

Using the forward and backward variables, this can be expressed as

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{P(Y|\ \lambda)} \tag{2.35}$$

This problem can be solved efficiently with the *Viterbi algorithm*, which again employs Dynamic Programming.

2.4.3 Support Vector Machines

Support Vector Machines (SVMs) are another type of supervised machine learning models, which are able to learn relationships between feature vectors $x_i \in \mathbb{R}^n$ and their expected classes $y_i, i = 1, ..., L$. SVMs attempt to solve this problem by grouping the training data vectors x_i and finding separating (hyper-)planes (with the normal vector w and the offset b) between the points of the different classes (or annotation labels) y_i. In doing so, they try to maximize the margin between the plane and the data points. Additionally, the feature vectors may be transformed into a higher-dimensional

space by the function $\phi(x_i)$ to make them more easily separable (as demonstrated in figure 2.17).

In [57], this training process is expressed (for a two-class problem) as:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^\top w + C \sum_{i=1}^{L} \xi_i \\ \text{subject to} \quad & y_i(w^\top \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \end{aligned} \tag{2.36}$$

(with ξ_i being a slack variable and $C > 0$ being a penalty parameter for the error term, higher Cs allowing for fewer outliers ([58]).

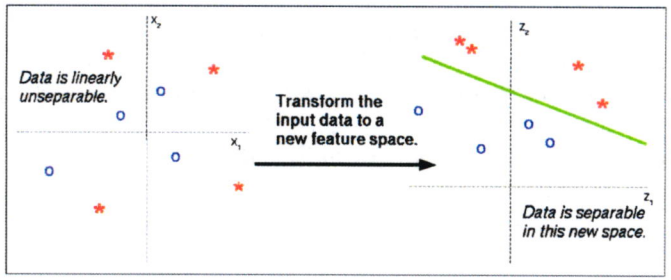


Figure 2.17: A set of data points which cannot be separated linearly in their original form (left), but can be separated after transformation into another space (right). [58]

$K(x_i, x_j) \equiv \phi(x_i)^\top \phi(x_j)$ is called a "kernel function". Several variants are possible, e.g. a linear kernel:

$$K(x_i, x_j) = x_i^\top x_j \tag{2.37}$$

It is often useful to use a non-linear kernel because the data points may not be linearly separable (even after the transformation into a higher-dimensional space). An example is shown in figure 2.18. The Radial Basis Function (RBF) kernel is a popular one:

$$K(x_i, x_j) = e^{-\gamma |x_i - x_j|^2}, \gamma > 0 \tag{2.38}$$

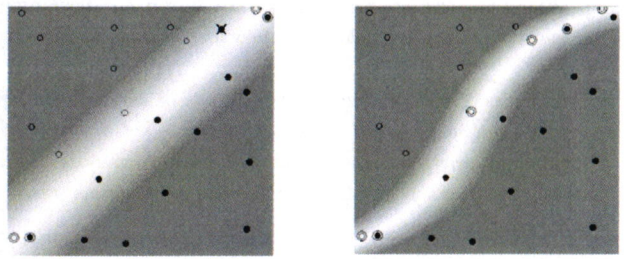


Figure 2.18: A set of data points which cannot be separated using a linear kernel (left), but can be separated with a polynomial kernel (right). [59]

As can be seen from the above equations, C and γ are free parameters. Their optimum values depend on the actual training data vectors. In [57], a grid search during each training is suggested to find them.

The presented training process is useful for solving two-class problems. For multi-class problems, one-vs-one trainings for all combinations of classes are performed. Then, all of the developed classifiers are used for the classification of the evaluation data and a voting strategy is applied to determine the resulting class.

2.4.4 i-Vector processing

i-Vector (identity vector) extraction was first introduced in [60], and has since become a state-of-the-art technique for various speech processing tasks, such as speaker verification, speaker recognition, and language identification [61]. i-Vector extraction is not a stand-alone training algorithm, but rather a feature post-processing step using unsupervised machine learning. The resulting i-vectors for training examples are used to train other models (instead of the features themselves); during classification, i-vectors are extracted in the same way, and run through this model.

The main idea behind i-vectors is that all training examples (e.g. speech utterances) contain some common trends, which effectively add irrelevance to the data during training. Using i-vector extraction, this irrelevance can be filtered out, while only the unique parts of the data relevant to the task at hand remain. The dimensionality of the training data is massively reduced, which also makes the training less computationally expensive. As a side effect, all feature matrices are transformed into i-vectors of equal

length, eliminating problems that are caused by varying utterance lengths. Mathematically, this assumption can be expressed as:

$$M(u) = m + Tw \tag{2.39}$$

where $M(u)$ is the GMM supervector for utterance u. The supervector approach was first presented in [62] and has since been successfully applied to a number of speech recognition problems. A music example can be found in [63]. m represents the language- and channel-independent component of u and is estimated using a Universal Background Model (UBM). T is a low-rank matrix modeling the relevant language- and channel-related variability, the so-called Total Variability Matrix. Finally, w is a normally distributed latent variable vector: The i-vector for utterance u.

The following steps are necessary for i-vector extraction:

1. **UBM training** A Universal Background Model (UBM) is trained using Gaussian Mixture Models (GMMs) from all utterances. This unsupervised model represents the characteristics that are common to all of them.

2. **Statistics extraction** 0th and 1st order Baum-Welch statistics are calculated for each of the utterances from the UBM according to:

$$N_c(u) = \sum_{t=1}^{L} P(c|y_t, \Omega) \tag{2.40}$$

$$\tilde{F}_c(u) = \sum_{t=1}^{L} P(c|y_t, \Omega)(y_t - m_c) \tag{2.41}$$

   where $u = y_1, y_2, ..., y_L$ denotes an utterance with L frames, $c = 1, ..., C$ denotes the index of the Gaussian component, Ω denotes the UBM, m_c is the mean of the UBM mixture component c, and $P(c|y_t, \Omega)$ denotes the posterior probability that the frame y_t was generated by mixture component c. As the equation shows, the 1st order statistics are centered around the mean of each mixture component.

3. T **matrix training** Using the Baum-Welch statistics for all utterances, the Total Variability Matrix T is now trained iteratively according to:

$$w = (I + T^\top \Sigma^{-1} N(u) T)^{-1} T^\top \Sigma^{-1} \tilde{F}(u) \tag{2.42}$$

   using the Expectation Maximization algorithm.

4. **Actual i-vector extraction** Finally, an i-vector w can be extracted for each utterance using equation 2.42 again. This can also be done for unseen utterances, using a previously trained T, and in this way be used during classification.

2.4.5 Artificial Neural Networks

Artificial Neural Networks have a long research history. Based on an idea by McCulloch and Pitts from 1943 [64], they were slowly developed into functional algorithms for classification and pattern recognition. Rosenblatt proposed a hardware design for a single-layer perceptron in 1958 [65], while the first multi-layer networks were introduced by Ivakhnenko and Lapa in 1965 [66]. In 1969, Minsky and Papert posited many practical limitations for Neural Networks [67], which led to a decrease in interest.
The introduction of the backpropagation algorithm solved some of these issues and increased the training speed of multilayer networks [68], leading to a wider usage in speech recognition [69] and other fields, such as computer vision [70] and Natural Language Processing [71]. However, other algorithms such as SVMs began to produce better results over time and thus overtook Neural Networks in popularity.
Over time, processing speed of computers increased, and better strategies and hardware for parallel computing became available. This the training of networks with many more layers possible, allowing for a much better adaptation to high-dimensional problems [72]. Over the past 10 years, this so-called "deep learning" became the state of the art for many machine learning problems [73][36][74].

Artifical Neural Networks (ANNs) are inspired by "real" Neural Networks - i.e. the human brain and nervous system. They consist of neurons, which are nodes that can process inputs and send outputs, and the connections between them. These neurons are grouped in layers: An input layer, an output layer, and a number of hidden layers in between them. Historically, ANNs had no hidden layers at all; this type of ANN was called "perceptron". Later, hidden layers were introduced and the resulting networks were called "Multilayer Perceptrons" (MLPs). Networks with no hidden layers are only able to solve linear problems; the introduction of hidden layers added non-linear projections to the calculation. Recent "Deep" Neural Networks (DNNs) possess three or more hidden layers, leading to an exponential increase of the degrees of freedom and thus the possibility to model much more complex problems.

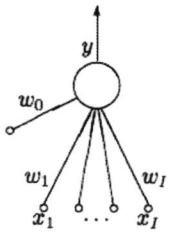
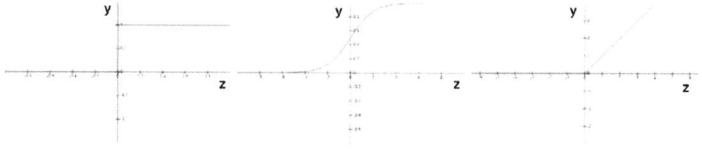


Figure 2.19: Functionality of a single neuron in an ANN. The neuron computes a weighted sum of its inputs, and then applies an activation function, resulting in output y. [75]



(a) Heaviside binary step (b) Sigmoid (c) ReLU

Figure 2.20: Activation functions used for ANN neurons. [76]

The function of a single neuron is visualized in figure 2.19. classic neurons compute a weighted sum z of their inputs $x = (x_1, x_2, ..., x_I)$:

$$z = w_0 + \sum_{i=1}^{I} x_i w_i = w_0 + w^\top x \tag{2.43}$$

where $w = (w_1, w_2, ..., w_I)$ is a vector of weights, and w_0 is a constant bias. This result is often called the "logit". Then, a nonlinear activation function can be applied. In perceptrons, this was the Heaviside step function, shown in figure 2.20a, resulting in a binary output:

$$y = \begin{cases} 1 \text{ if } z \leq 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.44}$$

An activation function commonly used is the sigmoid function, shown in figure 2.20b:

$$y = \frac{1}{1 + e^{-z}} \tag{2.45}$$

Neurons of this type are called "logistic" neurons. This function is often applied because it generates a smooth, real-valued output and has easy-to-use derivatives, simplifying the training process.
Rectified linear units (ReLUs) are also used frequently since they train faster than logistic units and retain more information that is relevant in the middle layers (in particular, this is leads to sparsity for small inputs and a lower risk of vanishing or exploding gradients). The function is shown in figure 2.20c:

$$y = \max(0, z) \tag{2.46}$$

In the last layer, a so-called softmax activation function is often applied. This function takes the outputs of all neurons into account, and computes a probability distribution (i.e. all the outputs will sum up to one and represent the likelihood of the corresponding class):

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{M} e^{z_k}}, 1 \leq j \leq M \tag{2.47}$$

where M is the number of output neurons.

In addition to the various types of neurons, ANNs themselves can be grouped into different of types. In their basic configuration, ANNs will have multiple layers of the described neurons where connections are only allowed in one direction. A schematic is

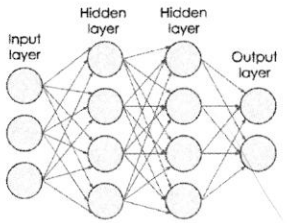


Figure 2.21: Schematic of a feed-forward neural network with an input layer with four neurons, two hidden layers with four neurons each, and an output layer with two neurons. [76]

shown in figure 2.21. This type of network is called "feed forward" or, in the context of Deep Learning, a Deep Neural Network (DNN).
If connections that skip back are allowed in the network, the network is called a Recurrent Neural Network (RNN). These networks are particularly useful for modeling time-dependent series because they have the ability to "store" temporal states in their units. However, they were deemed impractical for a long time because they exhibit the exploding/vanishing gradient problem during training [77]. This problem was solved with the introduction of memory cells in place of neurons, in particular Long Short-Term Memory (LSTM) units [78] and Gated Recurrent Units (GRU) [79]. Nowadays, RNNs are being used successfully in a number of research tasks [80].
A third type of Neural Network are Convolutional Neural Networks (CNNs). These networks add layers of filters (so-called convolutional layers) before or in between classic fully-connected layers. The parameters of these filters are trained jointly with the other layers. For this reason, CNNs are able to "learn" a feature representation of the input. They were first used in image recognition [81], but are now also being used for audio-related tasks such as environmental sound classification [82] and ASR [83].
A disadvantage of both RNNs and CNNs is the computational complexity of training them, and the requirement for even more training data because they possess even more degrees of freedom than DNNs of comparable sizes.
In this work, DNNs with logistic units in the hidden layers and a softmax output layer were used in phoneme classification tasks.

always?

Neural Network training is performed via the backpropagation algorithm. This algorithm is based on the calculation of a cost (or error) function E, which computes the difference between the network output and the expected output (e.g. the annotated

classes in the training data). Then, the partial derivatives of the cost are calculated with regards to the weights, using the outputs and logits for the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} \tag{2.48}$$

Then, the weights are adjusted accordingly:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{2.49}$$

where η is the so-called learning rate which must be chosen carefully to ensure convergence of the training process.
In the same way, the error is propagated further backwards through the model to adjust the weights of the previous layers. This is done until all the weights have been adjusted. Then, the next batch of training data is passed through the model and the process is repeated. The training process is performed until the weights converge (or until a fixed number of iterations has been reached).

A commonly used cost function is the squared error measure (employing the L^2 or Euclidean norm):

$$E_S = \frac{1}{2}\sum_{j=1}^{M}(t_j - y_j)^2 \tag{2.50}$$

where t_j is the target value for output unit j.

Alternatively, the cross-entropy is often chosen as the cost function when using a softmax output layer:

$$E_C = -\sum_{j=1}^{M} t_j \log y_j \tag{2.51}$$

One of the major issues in Neural Networks is that of overfitting: The model adapts too well to the training data and is not able to sufficiently generalize to new data anymore. (In extreme cases, the model simply learns all the training data by heart). This is especially relevant for deep networks because of the many degrees of freedom. There are several strategies to overcome this problem. One of the most frequently used regularization techniques is dropout: During training, nodes in the network are deactivated randomly, effectively resulting in many different network structures and making the whole network robust to variations in the data. The most crucial point, however, is the amount of data used for training the network. The more variety is

available, the lower the risk of overtraining.
The so-called "curse of dimensionality" is a concept that applies to many machine learning algorithms: However, Goodfellow et al. make an argument that Deep models operate on a different level altogether [84]: Where traditional machine learning methods make an assumption of smoothness of the hidden functions to be represented, deep models actually attempt to model the underlying structures in a distributed way. This means that information about outputs can be learned in a shared way - i.e., if two classes have something in common, the model is also able to represent this fact. Practically, the many degrees of freedom do not lead to "learning by heart", but instead allow for an internal representation of highly complex relationships. Goodfellow et al. mention two interpretations of the deep modeling capabilities: Learning a representation composed of simpler representations (e.g. corners defined by edges), or learning sequential step that build on top of each other (e.g. first locate objects, then segment them, then recognize them). There is also a number of publications that demonstrate better generalization abilities for deeper networks [85][86][87][88][81][89]. In an experiment in [90], shallow models with three hidden layers overfit at 20 million parameters, while a deep model (11 hidden layers) benefits from more than 60 million. This is because a deep model has the capability to learn the actual explanatory factors behind the expected outputs (e.g. learning about different genders or whether a person is wearing glasses when modeling faces [91]).

3 State of the Art

This chapter presents an overview over the various published approaches for the tasks considered in this work. One key issue is comparability: As in many MIR tasks, data sets are not publicly available and vary widely, making comparison impossible. As the following sections show, there is also disagreement about evaluation measures in all of these tasks.
For these reasons, new, reproducible data sets were created for this work (presented in the next chapter). As described in the previous chapter, the most frequently used measures were employed for evaluation.

3.1 From speech to singing

Singing presents a number of challenges for speech recognition when compared to pure speech [3][92][93]. The following factors make speech recognition on singing more difficult than on speech, and necessitate the adaption of existing algorithms.

Larger pitch fluctuations A singing voice varies its pitch to a much higher degree than a speaking voice. It often also has very different spectral properties.

Larger changes in loudness In addition to pitch, loudness also fluctuates much more in singing than in speech.

Higher pronunciation variation The musical context causes singers to pronounce certain sounds and words differently than if they were speaking them.

Larger time variations In singing, sounds are often prolonged for a certain amount of time to fit them to the music. Conversely, they can also be shortened or left out completely.
In order to research this effect more closely, a small experiment was performed on a speech corpus and on a singing data set (*TIMIT* and *ACAP*, see chapter 4): The standard deviations for all the phonemes in each data set were calculated.

The result is shown in figure 3.1, confirming that the variation in singing is much higher. This is particularly true for vowels.

Different vocabulary In musical lyrics, words and phrases often differ from normal conversational script. Certain words and phrases have different probabilities (e.g. a higher focus on emotional topics in singing).

Background music This is the biggest interfering factor with polyphonic recordings. Harmonic and percussive instruments add a big amount of spectral components to the signal, which lead to confusion in speech recognition algorithms. Ideally, these components should be removed or suppressed in a precursory step. This could be achieved, for example, by employing source separation algorithms. However, such algorithms add additional artifacts to the signal, and may not even be sufficient for this purpose at the current state of research.

Vocal activity detection (VAD) could be used as a non-invasive first step in order to discard segments of songs that do not contain singing voices. However, such algorithms often make mistakes in the same cases that are problematic for speech recognition algorithms (e.g. instrumental solos [94]).

For these reasons, most of the experiments in this work were performed on unaccompanied singing. The integration of the mentioned pre-processing algorithms would be a very interesting next step of research.

The lyrics-to-singing alignment algorithms presented in chapter 8 are an exception. Those were also tested on polyphonic music, and the algorithms appear to be largely robust to these influences.

3.2 Phoneme recognition

Due to the factors mentioned above in section 3.1, phoneme recognition on singing is more difficult than on clean speech. It has only been a topic of research for a few years, and there are few publications.

One of the earliest systems was presented by Wang et al. in 2003 [95]. Acoustic modeling is performed with triphone HMMs trained on read speech in Taiwanese and Mandarin. The language model is completely restricted to lines of lyrics in the test dataset. Testing is performed on 925 unaccompanied sung phrases in these language. Due to the highly specific language model, the word error rate is just 0.07.

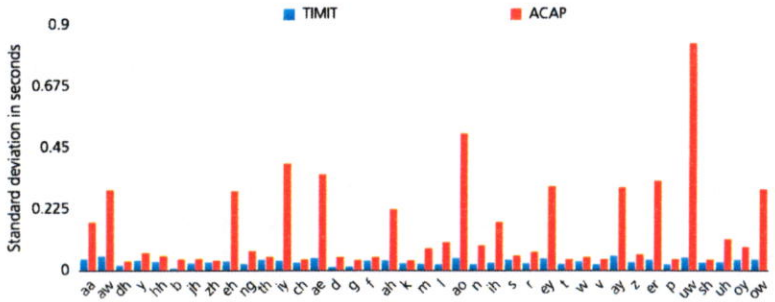


Figure 3.1: Standard deviations of phoneme durations in the *TIMIT* and *ACAP* data sets.

Hosoya et al. employ a similarly classic approach from ASR that employs monophone HMMs also trained on read speech for acoustic modeling [96] (2005). These models are adapted to singing voices using the Maximum Likelihood Linear Regression (MLLR) technique [97]. Language modeling is performed with a Finite State Automaton (FSA) specific to the Japanese language, making it more flexible than the previous system. The system is tested on five-word unaccompanied phrases, while the adaptation is performed on 127 choruses performed by different singers. The Word Error Rate is 0.36 without the adaptation, and 0.27 after adaptation.

In 2007, Gruhne et al. presented a classic approach that employs feature extraction and various machine learning algorithms to classify singing into 15 phoneme classes [98] [99]. The specialty of this approach lies in the pre-processing: At first, fundamental frequency estimation is performed on the audio input, using a Multi-Resolution Fast Fourier Transform (MRFFT) [100]. Based on the estimated fundamental frequency, the harmonic partials are retrieved from the spectrogram. Then, a sinusoidal re-synthesis is performed, using only the detected fundamental frequency and partials. Feature extraction is then performed on this re-synthesis instead of the original audio. Extracted features include MFCCs, PLPs, Linear Predictive Coding features (LPCs), and Warped Linear Predictive Coding features (WLPCs [101]). MLP, GMM, and SVM models are trained on the resulting feature vectors. The re-synthesis idea comes from a singer identification approach by Fujihara [102].

The approach is tested on more than 2000 separate, manually annotated phoneme instances from polyphonic recordings. Only one feature vector per phoneme instance

is calculated. Using SVM models, 56% of the tested instances were classified correctly into one of the 15 classes. This is significantly better than the best result without the re-synthesis step (34%).
In [103] (2010), the approach is expanded by testing a larger set of perceptually motivated features, and more classifiers. No significant improvements are found when using more intricate features, and the best-performing classifier remains SVM.

Fujihara et al. described an approach based on spectral analysis in 2009 [104]. The underlying idea is that spectra of polyphonic music can be viewed as the weighted sum of two types of spectra: One for the singing voice, and one for the background music. This approach then models these two spectra as probabilistic spectral templates. The singing voice is modeled by multiplying a vocal envelope template, which represents the spectral structure of the singing voice, with a harmonic filter, which represents the harmonic structure of the produced sound itself. This is analogous to the source-filter model of speech production [105]. For recognizing vowels, five such harmonic filters are prepared (`/a/ - /e/ - /i/ - /o/ - /u/`). Vocal envelope templates are trained on voice-only recordings, separated by gender. Templates for background music are trained on instrumental tracks. In order to recognize vowels, the probabilities for each of the five harmonic templates are estimated. As a side product, the algorithm also estimates the fundamental frequency of the singing voice.
As described, the phoneme models are gender-specific and only model five vowels, but also work for singing with instrumental accompaniment. The approach is tested on 10 Japanese-language songs. The best result is 65% correctly classified frames, compared to the 56% with the previous approach by this team, based on GMMs.

In 2009, Mesaros et al. also picked Hosoya's approach back up by using MFCC features and GMM-HMMs for acoustic modeling [106], and adapting the models for singing voices. These models are trained on the CMU ARCTIC speech corpus[1]. Then, different MLLR techniques for adapting the models to singing voices are tested [97]. The adaptation and test corpus consists of 49 voice-only fragments from 12 pop songs with durations between 20 and 30 seconds. The best results are achieved when both the means and variances of the Gaussians are transformed with MLLR. The results improved slightly when not just a single transform was used for all phonemes, but when they were grouped into base classes beforehand, each receiving individual transformation parameters. The best result is around 0.79 Phoneme Error Rate on the test

[1] `http://festvox.org/cmu_arctic/`

set.
In [107] and [108], language modeling is added to the presented approach. Phoneme-level language models are trained on the CMU ARCTIC corpus as unigrams, bigrams, and trigrams, while word-level bigram and trigram models are trained on actual song lyrics in order to match the application case. The output from the acoustic models is then refined using these language models. The approach is tested on the clean singing corpus mentioned above, and on 100 manually selected fragments of 17 polyphonic pop songs. To facilitate recognition on polyphonic music, a vocal separation algorithm is introduced [109].
Using phoneme-level language modeling, the Phoneme Error Rate on clean singing is reduced to 0.7. On polyphonic music, it is 0.81. For the word recognition approach, the word error rate is 0.88 on clean singing, and 0.94 on the polyphonic tracks.
A more detailed voice adaptation strategy is tested in [110]. Instead of adapting the acoustic models with mixed-gender singing data, they are adapted gender-wise, or to specific singers. With the gender-specific adaptations, the average Phoneme Error Rate on clean singing is lowered to 0.81 without language modeling, and 0.67 with language modeling. Singer-specific adaptation does not improve the results, probably because of the very small amount of adaptation data in this case.
In [111] (2014), McVicar et al. build on a very similar baseline system, but also exploit repetitions of choruses to improve transcription accuracy. This has been done for other MIR tasks, such as chord recognition, beat tracking, and source separation. They propose three different strategies for combining individual results: Feature averaging, selection of the chorus instance with the highest likelihood, and combination using the Recogniser Output Voting Error Reduction (ROVER) algorithm [112]. They also employ three different language models, two of which were matched to the test songs (and therefore not representative for general application). 20 unaccompanied, English-language songs from the RWC database [113] were used for testing; chorus sections were selected manually. The best-instance selection and the ROVER strategies improve results significantly; with the ROVER approach and a general-purpose language model, the Phoneme Error Rate is at 0.74 (versus 0.76 in the baseline experiment), while the Word Error Rate is improved from 0.97 to 0.9. Interestingly, cases with a low baseline result benefit the most from exploiting repetition information.

The final system was proposed by Hansen in 2012 [27]. It also employs a classic approach consisting of a feature extraction step and a model training step. Extracted features are MFCCs and TRAP features. Then, MLPs are trained separately on both

feature sets. As mentioned in section 2.2.4, each feature models different properties of the considered phonemes: Short-term MFCCs are good at modeling the pitch-independent properties of stationary sounds, such as sonorants and fricatives. On the flip-side, TRAP features are able to model temporal developments in the spectrum, forming better representations for sounds like plosives or affricates.
The results of both MLP classifiers are combined via a fusion classifier, also an MLP. Then, Viterbi decoding is performed on its output.
The approach is trained and tested on a data set of 13 vocal tracks of pop songs, which were manually annotated with a set of 27 phonemes. The combined system achieves a recall of 0.48, compared to 0.45 and 0.42 for the individual MFCC and TRAP classifiers respectively. This confirms the assumption that the two features complement each other. The phoneme-wise results further corroborate this.

Various publications suggest that phoneme recognition is not a trivial task for human listeners either. Recognition rates are in the range of 70 to 90% for native speakers, depending on factors such the signal quality and the type of phoneme [114][115]. For singing, much lower rates are reported [116].

3.3 Lyrics-to-audio alignment

In contrast to the other tasks discussed in this chapter, the task of lyrics-to-audio alignment has been the focus of many more publications. A comprehensive overview until 2012 is given in [92].
A first approach was presented in 1999 by Loscos et al. [3]. The standard forced alignment approach from speech recognition is adapted to singing. MFCCs are extracted first, and then a left-to-right HMM is employed to perform alignment via Viterbi decoding. Some modifications are made to the Viterbi algorithm to allow for low-delay alignment. The approach is trained and tested on a very small (22 minutes) database of unaccompanied singing, but no quantitative results are given.
The first attempt to synchronize lyrics to polyphonic recordings was made by Wang et al. in 2004 [117]. They propose a system, named "LyricAlly", to provide line-level alignments for karaoke applications. Their approach is heavily based on musical structure analysis. First, the hierarchical rhythm structure of the song is estimated. The result is combined with an analysis of the chords and then used to split the song into sections by applying a chorus detection algorithm. Second, Vocal Activity Detection (VAD) using HMMs is performed on each section. Then, sections of the text lyrics are

assigned to the detected sections (e.g. verses, choruses). In the next step, the algorithm determines whether the individual lines of the lyrics match up with the vocal sections detected by the VAD step. If they do not, grouping or partitioning is performed. This is based on the assumption that lyrics match up to rhythmic bars as determined by the hierarchical rhythm analysis. The expected duration of each section and line is estimated using Gaussian distributions of phoneme durations from a singing data set. In this manner, lines of text are aligned to the detected vocal segments. The approach is tested on 20 manually annotated pop songs. On the line level, the average error is 0.58 seconds for the starting points and −0.48 seconds for the durations. The system components are analyzed in more detail in [118].
In 2006, the same team presented an approach that also performs rhythm and bar analysis to facilitate syllable-level alignment [119]. For the phoneme recognition step, an acoustic model is trained on speech data and adapted to singing using the previously mentioned 20 songs. The possible syllable positions in the alignment step are constrained to the note segments detected in the rhythm analysis step. Due to annotator disagreement on the syllable level, the evaluation is performed on the word level. On three example songs, the average synchronization error rate is 0.19 when allowing for a tolerance of 1/4 bar.

Sasou et al. presented a signal parameter estimation method for singing employing an auto-regressive HMM (AR-HMM) in 2005 [120]. This method is particularly suited for modeling high-pitched signals, which is important for singing voices and usually not a focus in speech processing techniques. Models trained on speech are adapted to singing using MLLR. For evaluation, the method is applied to the task of lyrics-to-audio alignment and tested on 12 Japanese-language songs. For each song, a specific language model is prepared. The correct word rate is 0.79.

Chen et al. presented an approach based on MFCC features and Viterbi alignment in 2006 [121]. Vocal Activity Detection is performed as a pre-processing step, and then GMM-HMMs are used for Viterbi alignment between the audio and the lyrics. Once again, MLLR is used to adapt the acoustic models to singing. In addition, the grammar is specifically tailored to the lyrics. On a data set of Chinese songs by three singers, a boundary accuracy of 0.76 is obtained on the syllable level.

A similar approach which does not require in-depth music analysis was presented by Fujihara et al. in 2006 [122]. Once again, a straightforward Viterbi alignment method

from speech recognition is refined by introducing three singing-specific pre-processing steps: Accompaniment sound reduction, Vocal Activity Detection, and phoneme model adaptation.
For accompaniment reduction, the previously mentioned harmonic re-synthesis algorithm from [102] is used. For Vocal Activity Detection, a HMM is trained on a small set of unaccompanied singing using LPC-derived MFCCs and fundamental frequency (F_0) differences as features. The HMM can be parameterized to control the rejection rate.
For the phoneme model adaptation, three consecutive steps are tested: Adaptation to a clean singing voice, adaptation to a singing voice segregated with the accompaniment reduction method, and on-the-fly adaptation to a specific singer. MFCC features are used for the Viterbi alignment, which is performed on the vowels and syllabic nasals (`/m/, /n/, /l/`) only.
Ten Japanese pop songs were used for testing. Evaluation was done one the phrase level by calculating the proportion of the duration of correctly aligned sections to the total duration of the song. For eight of the ten songs, this proportion was 0.9 or higher when using the complete system, which the authors judge as satisfactory. Generally, the results are lower for performances by female singers, possibly because of the higher F_0s. These performances also benefit the most from the Vocal Activity Detection step, even though its performance is also somewhat worse for female singing. All three levels of phoneme model adaptations contribute to the success of the approach.
In 2008, the authors improved upon this system with three modifications: Fricative detection, filler models, and new features for the Vocal Activity Detection step [123]. Fricative detection is introduced because the previous system was only based on vowels and nasals, due to the fact that the harmonic re-synthesis discards other consonants. In the new system, fricatives are detected before this step and then retained for the alignment (stops are not used because they are too short).
The filler model is employed because singers sometimes add extraneous lyrics (like "la la la" or "yeah") to their performances.
As mentioned above, Vocal Activity Detection does not work as well for female performances because of inaccuracies in the spectral envelope estimation in high pitch regions. For this reason, the features are replaced in the new version by comparing the power of the harmonic components directly with those of similar F_0 regions.
The approach is again evaluated on ten Japanese pop songs. The original system produces an average accuracy of 0.81, which is raised to 0.85 with the new improvements.
In [124], the whole system is presented succinctly and evaluated in more detail. Additionally, integration into a music playback interface is described.

Mauch et al. augmented the same approach in 2010 by using chord labels, which are often available in combination with the lyrics on the internet [125][126]. Chords usually have longer durations than individual phonemes, and are therefore easier to detect. In this way, they provide a coarse alignment, which can be used to simplify the shorter-scale phoneme-level alignment. A chroma-based approach is used to estimate the chords. Information about the chord alignments is directly integrated into the HMM used for alignment.
In [126], a large range of parameterizations is tested on 20 English-language pop songs. The highest accuracy for the baseline approach (without chord information) is 0.46. Using chord position information, this rises to 0.88. Interestingly, Vocal Activity Detection improves the result when not using chords, but decreases it for the version with chord alignments, possibly because the coarse segmentation it provides is already covered by the chord detection in the second case. The method is also able to cope with incomplete chord transcriptions while still producing satisfactory results.

In 2007, Wong et al. presented a specialized approach for Cantonese singing that does not require phoneme recognition [127]. Since Cantonese is a tonal language, prosodic information can be inferred from the lyrics. This is done by estimating relative pitch and timing of the syllables by using linguistic rules. On the other side, a vocal enhancement algorithm is applied to the input signal, and its pitches and onsets are calculated. Then, both sets of features are aligned using DTW.
14 polyphonic songs were used for evaluation. The approach reaches an average (duration) accuracy of 0.75.

Lee et al. also follow an approach without phoneme recognition [128] (2008). It is purely based on structural analysis of the song recording, which is performed by calculating a self-similarity matrix and using it for segmentation. The algorithm takes structural a-priori knowledge into account, e.g. the fact that choruses usually occur most frequently and do not differ much internally. Lyrics segments are annotated by hand, splitting them up into paragraphs and labeling them with structural part tags ("intro", "verse", "chorus", and "bridge"). Then, Dynamic Programming is performed to match the lyrics paragraphs to the detected musical segments. A Vocal Activity Detection step is also introduced. Testing the approach on 15 English-language pop songs with 174 lyrics paragraphs in total, they obtain an average displacement error

of 3.5 seconds.

Mesaros et al. also present an alignment approach that makes use of their phoneme recognition approach described above in 3.2 [129], adding a harmonic re-synthesis step for vocal separation. Based on these models, they employ Viterbi alignment and obtain an average displacement error of 1.4 seconds for line-wise alignment (0.12 seconds when not using absolute differences). The test set consists of 17 English-language pop songs. They identify mistakes in the vocal separation step as the main source of error.

Two specialized approaches were presented in the past two years. The first one is part of a score-following algorithm by Gong et al. [130]. Here, vowels are used to aid alignment of the musical score to the recording, assuming the score contains the lyrics. This is done by training vowel templates on a small set of sung vowels. Spectral envelopes are used as features. Two different strategies for fusing the vowel and melody information are tested ("early" and "late"), as well as singer-specific adaptation of the templates via Maximum A-Posteriori (MAP) estimation. The training set consists of 160 vowel instances per singer, the test set of 8 full unaccompanied French-language songs per singer. The average displacement error is around 68ms for both singer-specific and -adapted models (best strategy).

Finally, Dzhambazov et al. presented a method that integrates knowledge of note onsets into the alignment algorithm [131]. Pitch extraction and note segmentation are performed in parallel with phoneme recognition via HMMs, and both results are refined with a transition model. A variable time HMM (VTHMM) is used to model the rules for phoneme transitions at note onsets.
On a test dataset of 12 unaccompanied Turkish-language Makam performances, the method achieves an alignment accuracy of 0.76. For polyphonic recordings (usually with accompaniment by one or more string instruments), a vocal re-synthesis step is introduced. The average accuracy in this case is 0.65.

3.4 Lyrics retrieval

As described in 3.2, Hosoya et al. developed a system for phoneme recognition, which they also apply to lyrics retrieval [96]. On a dataset of 238 children's songs, they obtain a retrieval rate of 0.86 for the Top 1 result, and of 0.91 for the Top 10 results. In [132] and [133], more experiments are conducted. As a starting point, the number of words

in the queries is fixed at 5, resulting in a retrieval rate of 0.9 (Top 1 result). Then, a melody recognition is used to verify the matches proposed by the speech recognition step, raising the retrieval rate to 0.93. The influence of the number of words in the query is also evaluated, confirming that retrieval becomes easier the longer the query is. However, even at a length of just three words, the retrieval rate is 0.87 (vs. 0.81 without melody verification).

Similarly, Wang et al. presented a query-by-singing system in 2010 [134]. The difference here is that melody and lyrics information are weighted equally in the distance calculation. Lyrics are recognized with a bigram HMM model trained on speech. The results are interpreted as syllables. A syllable similarity matrix is employed for calculating phoneme variety in the query, which is used for singing vs. humming discrimination. Assuming that only the beginning of each song is used as the starting point for queries, the first 30 syllables of each song are transformed into an Finite State Machine (FSM) language model and used for scoring queries against each song in the database. The algorithm is tested on a database of 2154 Mandarin-language songs, of which 23 were annotated and the remainder are used as "noise" songs. On the Top 1 result, a retrieval rate of 0.91 is achieved for the system combining melody and lyrics information, compared to 0.88 for the melody-only system.

As described in 3.2, Mesaros et al. developed a sophisticated system for phoneme and word recognition in singing. In [135], [110], and [108], they also describe how this system can be used for lyrics retrieval. This is the only purely lyrics-based system in literature. Retrieval is performed by recognizing words in queries with the full system, including language modeling, and then ranking each lyrics segment by the number of matching words (bag-of-words approach). The lyrics database is constructed from 149 song segments (lasting between 9 and 40 seconds in the corresponding recordings). Recordings of 49 of these segments are used as queries to test the system. The Top 1 retrieval rate is 0.57 (0.71 for the Top 10).

3.5 Language identification

A first approach for language identification in singing was proposed by Tsai and Wang in 2004 [136]. At its core, the algorithm is similar to Parallel Phone Recognition followed by Language Modeling (PPRLM). However, instead of full phoneme modeling, they employ an unsupervised clustering algorithm to the input feature data and tok-

enize the results to form language-specific codebooks (plus one for background music). Following this, the results from each codebook are run through matching language models to determine the likelihood that the segment was performed in this language. Prior to the whole process, Vocal Activity Detection is performed. This is done by training GMMs on segments of each language, and on non-vocal segments. MFCCs are used as features.
The approach is tested on 112 English- and Mandarin-language polyphonic songs each, with 32 of them being the same songs performed in both languages. A classification accuracy of 0.8 is achieved on the non-overlapping songs. On the overlapping songs, the accuracy is only 0.7, suggesting some influence of the musical material (as opposed to the actual language characteristics). Misclassifications occur more frequently on the English-language songs, possibly because of accents of Chinese singers performing in English, and because of louder background music.

A second, simpler approach was presented by Schwenninger et al. in 2006 [137]. They also extract MFCC features, and then use these to directly train statistical models for each language. Three different pre-processing strategies are tested: Vocal Activity Detection, distortion reduction, and azimuth discrimination. Vocal Activity Detection (or vocal/non-vocal segmentation) is performed by thresholding the energy in high-frequency bands as an indicator for voice presence over 1-second windows. This leaves a relatively small amount of material per song. Distortion reduction is employed to discard strong drum and bass frames where the vocal spectrum is masked by using a Mel-scale approach. Finally, azimuth discrimination attempts to detect and isolate singing voices panned to the center of the stereo scene.
The approach is tested on three small data sets of speech, unaccompanied singing, and polyphonic music. Without pre-processing steps, the accuracy is 0.84, 0.68, and 0.64 respectively, highlighting the increased difficulty of language identification on singing versus speech, and on polyphonic music versus pure vocals. On the polyphonic corpus, the pre-processing steps do not improve the result.

In 2011, Mehrabani and Hansen presented a full PPRLM approach for sung language identification. MFCC features are run through phoneme recognizers for Hindi, German, and Mandarin; then, the results are scored by individual language models for each considered language. In addition, a second system is employed which uses prosodic instead of phonetic tokenization. This is done by modeling pitch contours with Legendre polynomials, and then quantizing these vectors with previously trained

GMMs. The results are then again used as inputs to language models.
The approach is trained and tested on a corpus containing 12 hours of unaccompanied singing and speech in Mandarin, Hindi, and Farsi. The average accuracy for singing is 0.78 and 0.43 for the phoneme- and prosody-based systems respectively, and 0.83 for a combination of both.

Also in 2011, Chandrasekhar et al. presented a very interesting approach for language identification on music videos, analyzing both audio and video features [138]. On the audio side, the spectrogram, volume, MFCCs, and perceptually motivated Stabilized Auditory Images (SAI) are used as inputs. One-vs-all SVMs are trained for each language. The approach is trained and tested on 25,000 music videos in 25 languages. Using audio features only, the accuracy is 0.45; combined with video features, it rises to 0.48. It is interesting to note that European languages achieve much lower accuracies than Asian and Arabic ones. English, French, German, Spanish and Italian rank below 0.4, while languages like Nepali, Arabic, and Pashto achieve accuracies above 0.6. It is possible that the language characteristics of European languages make them harder to discriminate (especially against each other) than others.

3.6 Keyword spotting

Keyword spotting in singing was first attempted in 2008 by Fujihara et al. [139]. Their method starts with a phoneme recognition step, which is once again based on the vocal re-synthesis method described in [102]. MFCCs and power features are extracted from the re-synthesized singing and used as inputs to a phoneme model, similar to Gruhne's phoneme recognition approach mentioned above in 3.2. Three phoneme models are compared: One trained on pure speech and adapted with a small set of singing recordings, one adapted with all recordings, and one trained directly on singing. Viterbi decoding is then performed using keyword-filler HMMs to detect candidate segments where keywords may occur. These segments are then re-scored through the filler HMM to verify the occurrence.
The method is tested on 79 unaccompanied Japanese-language songs from the RWC database [113] with keywords containing at least 10 phonemes. The Phoneme Error Rate is 0.73 for the acoustic models trained on speech, 0.67 for the adapted models, and 0.49 for the models trained on singing (it should be mentioned that the same songs were used for training and testing, although a cross-validation experiment shows that the effect is negligible). The employed evaluation measure is "link success rate",

describing the percentage of detected phrases that were linked correctly to other occurrences of the phrase in the data set. In that sense, it is a sort of accuracy measure. The link success rate for detecting the keywords is 0.3. The authors show that the result depends highly on the number of phonemes in the considered keyword, with longer keywords being easier to detect.

In 2012, Mercado et al. presented an approach to keyword spotting in singing based on a different principle: DTW between a sung query and the requested phrase in the song recording. In particular, Statistical Sub-Sequence DTW is the algorithm employed for this purpose. MFCCs are used as feature inputs, then the costs of the warping paths are calculated from all possible starting points to obtain candidate segments, which are then further refined to find the most likely position.
The approach is tested on a set of vocal tracks of 19 pop songs (see section 4.2.2) as the references, and recordings of phrases sung by amateur singers as the queries, but no quantitative results are given. The disadvantage of this approach lies in the necessity for audio recordings of the key phrases, which need to have at least similar timing and pitch as the reference phrases.

Finally, Dzhambazov et al. developed a score-aided approach to keyword spotting in 2015 [140]. A user needs to select a keyword phrase and a single recording in which this phrase occurs. The keyword is then modeled acoustically by concatenating recordings of the constituent phonemes (so-called acoustic keyword spotting). Similar to Mercado's approach, Sub-Sequence DTW is performed between the acoustic template and all starting positions in the reference recording to obtain candidate segments. These segments are then refined by aligning the phonemes to the score in these positions to model their durations. This is implemented with Dynamic Bayesian Network HMMs. Then, Viterbi decoding is performed to re-score the candidate segments and obtain the best match.
The approach is tested on a small set of unaccompanied Turkish-language recordings of traditional Makam music. The Mean Average Precision (MAP) for the best match is 0.08 for the DTW approach only, and 0.05 for the combined approach. For the top-6 results, the MAP is 0.26 and 0.38 respectively.

4 Data Sets

This chapter contains descriptions of all the data sets (or corpora) used over the course of this thesis. They are grouped into speech-only data sets, data sets of unaccompanied (= a-capella) singing, and data sets of full musical pieces with singing ("real-world" data sets). The final section lists the keywords chosen for the keyword spotting tasks and describes their selection process.

4.1 Speech data sets

4.1.1 TIMIT

TIMIT is, presumably, the most widely used corpus in speech recognition research [141]. It was developed in 1993 and consists of 6,300 English-language audio recordings of 630 native speakers with annotations on the phoneme, word, and sentence levels. The corpus is split into a training and a test section, with the training section containing 4,620 utterances, and the test section containing 1,680. Each of those utterances has a duration of a few seconds. The recordings are sampled at 16,000Hz and have a mono channel.
The phoneme annotations contain 61 different phonemes and follow a model similar to ARPABET, a popular set of phonetic symbols developed by the Advanced Research Projects Agency (ARPA) [142]. In this work, the annotations are broken down to a set of 39 phonemes as suggested in [143]. This phoneme set is commonly used in speech recognition, e.g. by the CMU Sphinx framework, and is listed in appendix A.1.
As described in [142], the data was collected and annotated in a sophisticated process, and verified multiple times. It can therefore be assumed to be correct.

4.1.2 NIST Language identification corpus (NIST203LRE)

The National Institute for Standards and Technology (NIST) regularly runs various speech recognition challenges, one of them being the Language Recognition Evaluation

(LRE) task, which is held every two to four years[1]. To this end, they publish training and evaluation corpora of speech in several languages. They consist of short segments (up to 35 seconds) of free telephone speech by many different speakers. The recordings are mono channel with a sampling rate of 8000Hz.

The corpus for the 2003 challenge was used in this work for comparison of language identification algorithms on speech data [144]. Only the English-, German-, and Spanish-language subsets were selected because these languages are covered by the corresponding singing data set. To balance out the languages, 240 recordings were used for each of them, summing up to around 1 hour of material per language. This corpus will be referred to as *NIST2003LRE*.

Since the speakers were instructed to speak their native language, the language annotations can be assumed to be correct. However, the recording quality is not very high, owing to the telephone-based recording process. Additionally, some segments do not contain a large variety of words, either because they are short, because the speaker repeats a word over and over ("okay... okay..."), or because the speaker produces other vocal noises like laughing.

4.1.3 OGI Multi-Language Telephone Speech Corpus (OGIMultilang)

In 1992, the Oregon Institute for Science and Technology (OGI) also published a multilingual corpus of telephone recordings, called the OGI Multi-Language Telephone Speech Corpus, to facilitate multi-language ASR research. Just like the NIST corpora, it has become widely used for speech recognition tasks. Again, the English-, German-, and Spanish-language subsets were used in this work. They each consist of more than 1000 recordings per language of up to 50 seconds duration, making up a total of about three to five hours. In this work, the corpus is called *OGIMultilang*. In contrast to the NIST corpus, the recordings are somewhat less "clean" with regards to accents and background noise; the sampling rate is also 8000Hz on a mono channel. The language annotations were subjected to manual verification. Similar to the NIST corpus, the audio quality is relatively low, and recordings are sometimes not very varied.

In many experiments, languages were balanced out by randomly discarding utterances to obtain equal numbers. For experiments on longer recordings, results on individual utterances were aggregated for each speaker and also balanced down to the lowest

[1] https://www.nist.gov/itl/iad/mig/language-recognition

number, producing 118 documents per language (354 in sum).

hh:mm:ss ***#Utterances*** **#Speakers** **∅ Duration/Speaker**	**NIST2003LRE**	**OGIMultilang**	**YTAcap**
English	00:59:08 *240* - -	05:13:17 *1912* 200 00:01:34	08:04:25 *1975* 196 00:02:28
German	00:59:35 *240* - -	02:52:27 *1059* 118 00:01:28	04:18:57 *1052* 116 00:02:14
Spanish	00:59:44 *240* - -	03:05:45 *1151* 129 00:01:26	07:21:55 *1810* 187 00:02:43

Table 4.1: Amounts of data in the three used data sets: Sum duration, number of utteraces, number of speakers, and average duration per speaker.

4.2 Unaccompanied singing data sets

4.2.1 YouTube data set (YTAcap)

As opposed to the speech case, there are no standardized corpora for sung language identification. For the sung language identification experiments, files of unaccompanied singing were therefore extracted from *YouTube*[2] videos. This was done for three languages: English, German, and Spanish. The corpus consists of between 116 (258min) and 196 (480min) examples per language. These were mostly videos of amateur singers freely performing songs without accompaniment. Therefore, they are of highly varying quality and often contain background noise (the language annotations, however, can be assumed to be correct). The audio was downloaded in the natively provided quality, but then downsampled to 8000Hz and averaged to a mono channel for uniformity and for compatibility with the speech corpora for language identification. Most of the performers contributed only a single song, with just a few providing up to three. This

[2] http://www.youtube.com

was done to avoid effects where the classifier recognizes the singer's voice instead of the language.

Special attention was paid to musical style. Rap, opera singing, and other specific singing styles were excluded. All the songs performed in these videos were pop songs. Different musical styles can have a high impact on language classification results.In order to limit this influence as much as possible, recordings of pop music were selected instead of language-specific genres (such as Latin American music). This data set is named *YTAcap* in this work.

For many experiments, these songs were split up into segments of 10-20 seconds at silent points (3,156 "utterances" in sum). In most cases, these segments were then balanced for the languages by randomly discarding superfluous segments. In other experiments, a balanced set of the whole songs was used (i.e. 116 songs per language). An overview of the amounts of data in the three corpora for language identification is given in table 4.1.

4.2.2 Hansen's vocal track data set (ACAP)

This is one of the data sets used for keyword spotting and phoneme recognition. It was first presented in [27], and consists of the vocal tracks of 19 commercial English-language pop songs. They have studio quality with some post-processing applied (EQ, compression, reverb). Some of them contain choir singing. These 19 songs are split up into 920 clips that roughly represent lines in the song lyrics. The original audio quality is 44,100Hz on a mono channel; for compatibility with models trained on *TIMIT*, they were downsampled to 16,000Hz.

13 of the songs were annotated with time-aligned phonemes. The phoneme set is the one used in CMU Sphinx[3] and contains 39 phonemes (it is the same one used in the *TIMIT* annotations described above, and can be found in appendix A.1). All of the songs were annotated with word-level transcriptions. This is the only one of the singing data sets that has full manual phoneme annotations, which were transcribed by a single annotator and may contain minor errors. Apart from this, the annotations are largely reliable and are used as ground truth in this work.

This data set will be referred to as *ACAP*.

[3] `http://cmusphinx.sourceforge.net/`

4.2.3 DAMP data set

As described, Hansen's data set is very small and therefore not suited to training phoneme models for singing. As a much larger source of unaccompanied singing, the *DAMP* data set, which is freely available from Stanford University[4][145], was used. This data set contains more than 34,000 recordings of amateur singing of full songs with no background music, which were obtained from the *Smule Sing!* karaoke app. Each performance is labeled with metadata such as the gender of the singer, the region of origin, the song title, etc. The singers performed 301 English-language pop songs. The recordings have good sound quality with little background noise, but come from a lot of different recording conditions. They were originally provided in OGG format at a sampling rate of 22,050Hz (mono channel), but were also converted to wave format and downsampled to 16,000Hz for compatibility. A list of the contained songs can be found in appendix A.3.3.

No lyrics annotations are available for this data set, but the textual lyrics can be obtained from the *Smule Sing!* website[5]. These are, however, not aligned in any way. Such an alignment was performed automatically on the word and phoneme levels (see section 5.3).

The selection of songs is not balanced, with performances ranging between 21 and 2038 instances. Female performances are also much more frequent than male ones, and gender often plays a role when training and evaluating models. For these reasons, several subsets of the dataset were composed by hand.

In order to generate training data sets, the data was balanced by songs so that certain songs (and their phoneme distributions) would not be overrepresented. Since the least represented song in the original data set has 21 recordings, 20 to 23 recordings per song were chosen at random to generate a training data set, which was named *DampB*. For each song, as many different singers as possible were selected. Additionally, recordings with more "Loves" (user approval) were more likely to be selected (however, a large percentage of the original data set did not have such ratings). This resulted in a data set of 6,902 recordings.

This process was then repeated, this time only taking recordings by singers of one gender into account. In this way, data sets of recordings by female and male singers only were, named *DampF* and *DampM* respectively. Due to the gender split, there were fewer recordings of some of the songs available. Male singers in particular are underrepresented in the original data set. Therefore, *DampF* contains 6,564 recordings,

[4] `https://ccrma.stanford.edu/damp/`
[5] `http://www.smule.com/songs`

while *DampM* contains 4,534. These sizes are roughly in the same range as for the mixed-gender data set, which enables the comparison of models trained on all three.
These three data sets do not contain balanced amounts of phonemes; therefore, subsets of them were created where phoneme frames were discarded until they were balanced and a maximum of 250,000 frames per phoneme were left, where possible. These data set are named *DampBB*, *DampFB*, and *DampMB*, and they are about 4% the size of their respective source data sets.
Since all of these data sets are still much larger than the *TIMIT* speech corpus, another subset of similar size was created for comparison. On the basis of the balanced *DampBB* data set, phoneme instances were discarded until 60,000 frames per phoneme were left, resulting in the *DampBB_small* data set.
For testing new algorithms, two small test data sets were created from the original *DAMP* data set. These are, again, split by gender, and contain one recording per song, resulting in 301 recordings for the female data set and 300 for the male one (since there was one song with not enough available male recordings). They are called *DampTestF* and *DampTestM* respectively. For mixed-gender training, testing was simply performed on both data sets.
Additionally, 20 phrases from each of the test data sets were manually selected for good singing and recording quality. This was necessary for fine-tuning and analysis of the retrieval algorithms.

To sum up:

DampB Contains 20 to 23 full recordings per song (more than 6000 in sum), both male and female.

DampBB Same as before, but phoneme frames were discarded until they were balanced and a maximum of 250,000 frames per phoneme were left, where possible. This data set is about 4% the size of *DampB*.

DampBB_small Same as before, but phoneme frames were discarded until they were balanced and 60,000 frames per phoneme were left (a bit fewer than the amount contained in *TIMIT*). This data set is about half the size of *DampBB*.

DampF and DampM Each of these data sets contains 20 to 23 full recordings per song performed by singers of one gender, female and male respectively.

DampFB and DampMB Starting from *DampF* and *DampM*, these data sets were

Name (#Utterances) *dd:hh:mm:ss*	Both genders	Female	Male
Full	DampB (6,902) *11:08:34:30*	DampF (6,654) *10:19:05:27*	DampM (4,534) *07:13:01:06*
Balanced	DampBB *08:49:02*	DampFB *08:29:39*	DampMB *05:53:01*
Reduced	DampBB_small *04:10:13*		
Test		DampTestF (301) *18:15:43*	DampTestM (300) *17:58:02*
Retrieval		DampRetrievalF (20) *01:56*	DampRetrievalM (20) *01:52*

Table 4.2: Overview of the structure of the *DAMP*-based phonetically annotated data sets, number of recordings in brackets, duration in italics (dd:hh:mm:ss). Note that no number of recordings is given for some data sets because their content was selected phoneme-wise, not song-wise. For retrieval, the recordings are short phrases instead of full songs.

then reduced in the same way as *DampBB*. *DampFB* is roughly the same size, *DampMB* is a bit smaller because there are fewer male recordings.

DampTestF and DampTestM Contains one full recording per song and gender (300 each). These data sets were used for testing. There is no overlap with any of the training data sets.

DampRetrievalF and DampRetrievalM 20 hand-picked sung phrases from *DampF* and *DampM* of a few seconds duration with good enunciation and good audio quality. These were selected for fine-tuning retrieval approaches. An overview can be found in appendix A.3.4.

An overview of the amounts of data is given in table 4.2.

4.2.4 Retrieval data set by the author (AuthorRetrieval)

This is a data set of 90 short lyrics phrases from the *DAMP* data set sung with clear enunciation by the author. The phrases were recorded with a smartphone microphone

for testing a demo app for the retrieval algorithm. They were then further utilized to finetune this retrieval algorithm. The recordings are a few seconds in duration with a sampling rate of 16,000Hz on a mono channel.

4.3 Accompanied singing data sets

4.3.1 QMUL Expletive data set

This data set consists of 80 popular full songs which were collected at Queen Mary University of London, most of them Hip Hop. 711 instances of 48 expletives were annotated on these songs. In addition, the matching textual, unaligned lyrics were retrieved from the internet. The audio is provided at 44,100kHz sampling rate in stereo, but was also downsampled to 16,000Hz and averaged to a mono track for compatibility with other models.

4.3.2 Mauch's data set

This data set was first presented in [126] and is used for alignment evaluation. It consists of 20 English-language pop songs with singing and accompaniment. Word onsets were manually annotated. This data set will be referred to as *Mauch*.

4.3.3 Hansen's vocal track data set (polyphonic) (ACAP_Poly)

This data sets consists of the songs in the *ACAP* data set described above in section 4.2.2, but with instrumental accompaniment. The annotations are carried over. This data set is also used for alignment evaluation and denoted as *ACAP_Poly*.

4.4 Keywords

From the 301 different song lyrics of the *DAMP* data sets, 15 keywords were chosen by semantic content and frequency to test the keyword spotting algorithms. Each keyword occurs in at least 50 of the 301 songs, and also appears in the *ACAP* data set. The keywords are shown in table 4.3. A list of the number of occurrences in each data set is given in appendix A.2.

#Phonemes	Keywords
2	eyes
3	love, away, time, life, night
4	never, baby, world, think, heart, only, every
5	always, little

Table 4.3: All 15 tested keywords, ordered by number of phonemes.

5 Singing Phoneme Recognition

In common ASR systems, phoneme recognition is performed by training an acoustic model that is able to determine the probability of phoneme occurrences in unseen audio frames. The result of this is a so-called phoneme posteriorgram - i.e. a matrix of the probabilities of each possible phoneme over time. In a second step, another model is applied to this posteriorgram, taking into account the probabilities of phoneme sequences and words formed by these phonemes, and thus generating a phoneme string from the posterior probabilities. This step is called language modeling.

Language modeling is a field of research unto itself, and is a complex tasks for lyrics. For this reason, this work focuses on improving the acoustic modeling step. The training process is performed according to the general schema described in section 2.1. The specific adaptations are shown in figures 5.1 and 5.2: In training, phoneme annotations are used to defined the possible classes. A set of around 39 phonemes is common; the phoneme set used in this work is the one used in CMU Sphinx[1] and is listed with examples in appendix A.1. The used data sets contain annotations of monophones. In ASR, splitting phonemes into three temporal phases is common, resulting in so-called senones. This was also tested as described in section 5.3.

The whole training process then results in an acoustic model, which can be used for classification as shown in figure 5.2. As described, classification results in a phoneme

[1] `http://cmusphinx.sourceforge.net/`

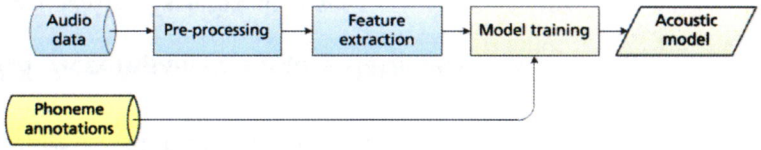


Figure 5.1: Schematic of the training procedure for phoneme recognition.

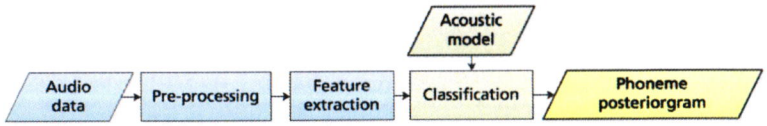


Figure 5.2: Schematic of the classification procedure for phoneme recognition.

posteriorgram. An example is shown in figure 5.3. The time resolution of the posteriograms in this work is 10ms. Instead of language modeling, many of the following tasks build directly on the posteriogram. For evaluation of the phoneme recognition itself, Viterbi decoding with evenly weighted transitions is performed on the posteriorgrams. Then, the Phoneme Error Rate (PER) and the Weighted Phoneme Error Rate (WPER) are used for assessing the quality (see section 2.3.2).

The following chapter describes the systems developed for phoneme recognition in singing, which serve as a basis for most of the following tasks. All three approaches are based on this general processing chain. The first one employs traditional GMM-HMM and DNN models trained on speech as a starting point. The other two use DNNs for modeling the phonemes, and are trained on different data sets. Since there is no large data set of singing annotated with phonemes available, speech was made more "song-like" for the second approach. In the third one presented, lyrics are automatically aligned to a large singing data set using the GMM-HMM approach, and the resulting data set is used for model training again.
The approaches are tested on the *ACAP* data set, where the results are compared to the manual annotations, and on the *DampTestF* and *DampTestM* data sets, which have automatically generated annotations, as described later in this section. Some experiments were also performed on the test portion of the *TIMIT* speech corpus. Cross validation was not necessary since there are is enough dedicated test data available for this task; this provides a clear separation between training and test data.

5.1 Phoneme recognition using models trained on speech

As a starting point, models for phoneme recognition were trained on speech data, specifically the *TIMIT* corpus. As in many classic ASR approaches, GMM-HMMs

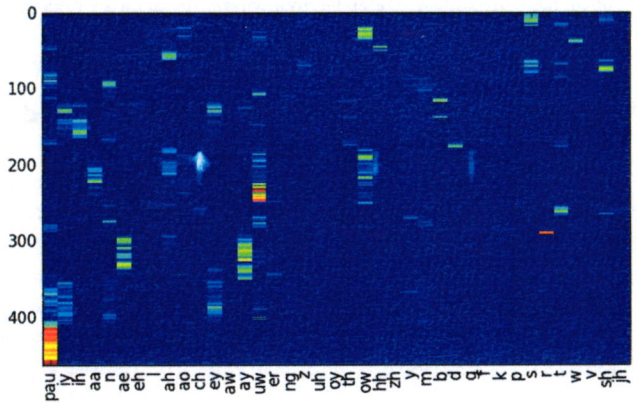


Figure 5.3: Example of a phoneme posteriorgram, the result of classification with an acoustic model. The posteriorgram represents the posterior probabilities of each phoneme over time. The time resolution in this example is 10ms.

were selected as a basis and trained using the Hidden Markov Toolkit (HTK)[146]. Additionally, Deep Neural Networks (DNN) were trained for comparison with models trained on the other data sets. They had three hidden layers of 1024 nodes, 850 nodes, and 1024 nodes again.

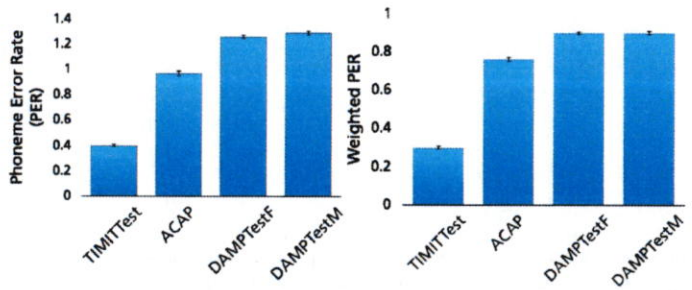
In both cases, 13 MFCCs were extracted, and deltas and double-deltas were calculated, resulting in a feature vector of dimension 39. As the output, the described set of 39 monophones was used.

The HMMs were used for aligning text lyrics to audio in some of the following approaches. To verify the quality of such an alignment, this was tested on the part of the *ACAP* singing corpus that has phoneme annotations. On average, the alignment error was 0.16 seconds. A small manual check suggests that this value is in the range of annotator uncertainty. A closer inspection of the results also shows that the biggest contributions to this error occur because a few segments were heavily misaligned, whereas most of them are just slightly shifted.

The DNN models were trained to perform phoneme recognition using the Theano framework [147]. This system was evaluated by first generating phoneme posteriorgrams from the test audio using the DNN models, and then running Viterbi decoding on those to extract phoneme strings. Then, the Phoneme Error Rate and the Weighted Phoneme Error Rate were calculated as described above.

The results for DNN models trained on *TIMIT* are shown in figure 5.4. For validation, the model was tested on the test part of *TIMIT*, resulting in a Phoneme Error Rate of 0.4 and a Weighted Phoneme Error Rate of 0.3. Lower values on these corpora can be found in literature, but those systems are usually more sophisticated and include language modeling steps and gender- or speaker-adapted models. In this scenario, those values serve as a validation of the recognition ability of the model on unseen data, and as an upper bound for the other results.

On the *ACAP* data set, the Phoneme Error Rate is 0.97 and the Weighted Phoneme Error rate is 0.76. Performing the same evaluation on the *DAMP* test data sets generates even worse results: A Phoneme Error Rate of 1.26/1.29 (female/male), and a Weighted Phoneme Error Rate of 0.9. This demonstrates that the performance of models trained on speech leaves room for improvement when used for phoneme recognition in singing. The difference between both singing data sets can be explained by their content: *ACAP* is much smaller, and contains cleaner singing in the sense of both the recording quality and the singing performance. Songs in this data set were performed by professional singers, whereas the *DAMPTest* sets contain recordings by amateurs who do not always enunciate clearly. Additionally, the phoneme annotations for the



(a) Phoneme Error Rate (b) Weighted Phoneme Error Rate

Figure 5.4: Mean phoneme recognition results on the test data sets using acoustic models trained on *TIMIT* (error bars represent standard error over utterances).

DAMPTest sets were generated from the song lyrics; these do not correspond to the actual performed phonemes in all cases.
It can be assumed that models trained on better-matching conditions (i.e. singing) would perform much better at this task. The problem with this approach lies in the lack of data sets that can be used for these purposes. In contrast with speech, no large corpora of phonetically annotated singing are available. In the following sections, workarounds for this problem are tested.

5.2 Phoneme recognition using models trained on "songified" speech

When there is a scarcity of suitable training data, attempts are often made to generate such data artificially from existing data for other conditions. For example, this is often done when models for noisy speech are required [148][149]. Similarly, so-called data augmentation methods are often employed to obtain more training data for machine learning models [150][151]. Inspired by this, one idea was making existing speech corpora more "song-like" and use these modified data sets to train models for phoneme recognition in singing. The *TIMIT* corpus was once again used as a basis for this.

An overview of the approach is shown in figure 5.5. Five variants of the training part of *TIMIT* are generated. MFCC features are then extracted from these new data sets and used to train models.

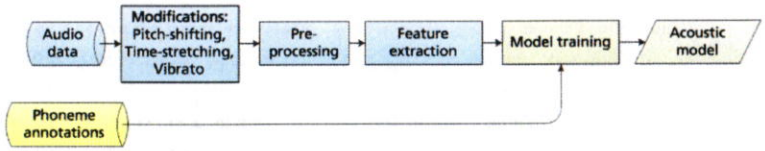


Figure 5.5: Overview of the "songified" phoneme recognition training process.

Similarly, MFCCs are extracted from the *TIMIT* test set and from the *ACAP* data set. The previously trained models are used to recognize phonemes on these test datasets. Phoneme sequences can be generated from the results with Viterbi decoding. Finally, the results are evaluated.

Several variants of making the training data more "song-like" were tested. Table 5.1 shows an overview over the five data sets generated from *TIMIT* using three modifications. Dataset N is the original *TIMIT* training set. For dataset P, four of the eight blocks of *TIMIT* were pitch-shifted. For dataset T, five blocks were time-stretched and vibrato was applied to two of them. In dataset TP, the same is done, except with additional pitch-shifting. Finally, dataset M contains a mix of these modified blocks. In detail, the modifications were performed in the following way:

Time stretching For time stretching, the phase vocoder from [152], which is an implementation of the Flanagan-Dolson phase vocoder [153][154], is used. This algorithm works by first performing a Short-Term Fourier Transform (STFT) on the signal, and then resampling the frames to a different duration and performing the inverse Fourier transform.
As described in section 3.1, time variations in singing mainly affect vowels and durations are often much longer than in speech. Therefore, the *TIMIT* annotations are used to only pick out the vowel segments from the utterances. They are randomly modified to a duration between 5 and 100 times the original duration, and then re-inserted into the utterance. This effectively leads to more vowel frames in the training data, but since there is already a large amount of instances for each phoneme in the original training data, the effects of this imbalance should be negligible.

Pitch shifting To pitch-shift the signal, code from the freely available Matlab tool *AutoTune Toy* [155], which also implements a phase vocoder, is used. In this implementation, the fundamental frequency is first detected automatically. The

	N	P	T	TP	M
DR1	N	N	N	N	N
DR2	N	N	N	N	N
DR3	N	N	N	N	P
DR4	N	N	T	TP	TV
DR5	N	P	T	TP	TPV
DR6	N	P	T	TP	TV
DR7	N	P	TV	TPV	P
DR8	N	P	TV	TPV	TPV

Table 5.1: The five *TIMIT* variants that were used for training (rows are blocks of the *TIMIT* training corpus, columns are the five generated datasets). Symbols: N - Unmodified; P - Pitch-shifted; T - Time-stretched; V - Vibrato

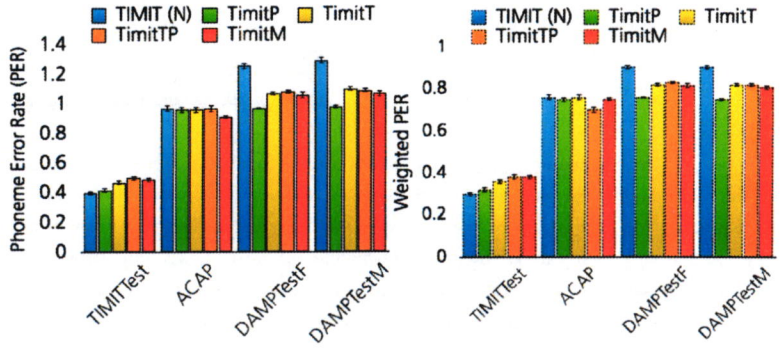
signal is then compressed or expanded to obtain the new pitch, and interpolated to retain the original duration.

Using the *TIMIT* annotations, utterances are split up into individual words, and then a pitch-shifted version of each word is generated and the results are concatenated. Pitches are randomly selected from a range between 60% and 120% of the original pitch.

Vibrato The code for vibrato generation was also taken from *AutoTune Toy*. It functions by generating a sine curve and using this as the trajectory for the pitch shifting algorithm mentioned above. A sine of amplitude 0.2 and frequency 6Hz is used.

In singing, vibrato is commonly performed on long sounds, which are usually vowels. Since spoken vowels are usually short, vibrato cannot be perceived on them very well. Therefore, vibrato is only added on time-stretched vowels.

The approach was tested on the various test data sets - namely, the test part of *TIMIT*, the *ACAP* data set, and the two test sets selected from the *DAMP* data set. Figure 5.6 shows the results for the DNN models. As it demonstrates, results for singing are generally worse than for speech. The base result for singing is a Weighted Phoneme Error Rate of 0.8 for *ACAP*, and of 0.9 for both *DAMPTestF* and *DAMPTestM* (with a model trained on the original *TIMIT* data set - also see previous section). When comparing the models trained on the various *TIMIT* modifications, an improvement is observed for all variants. In contrast, none of the modifications improve the result on the speech data at all. The base result here is 0.3. This makes sense since all of the modifications make the training data less similar to the test data set.



(a) Phoneme Error Rate (b) Weighted Phoneme Error Rate

Figure 5.6: Mean phoneme recognition results on the test data sets using acoustic models trained on *TIMIT* and augmented versions thereof (error bars represent standard error over utterances).

On singing, the Weighted Phoneme Error rate falls by 0.1 to 0.15 when using models trained on the pitch-shifted data set (*TimitP*), and by up to 0.08 when using the time-stretched training data (*TimitT*). The improvements for the corpora with both improvements lie in between. Vibrato does not have a strong influence on the result. The higher error rates for *DAMPTest* can, again, be explained with the fact that this data set has more variation in audio and singing quality.

Pitch shifting might have a stronger effect on the recognition result because it is a stronger modification in the sense that it generates actual new feature values. In contrast, time stretching mostly generates new frames with values similar to the ones in the original data set (and only in between those). Additionally, pitch shifting may introduce more variety that is closer to sung sounds because singers do not usually shape long vowels just by stretching out their short versions.

Nevertheless, it is interesting to see that both pitch shifting and time stretching improve the result for sung phoneme recognition. This indicates that more variety in the training data is a step in the right direction. However, there is still a lot of room for improvement.

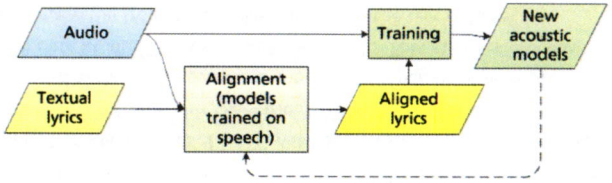


Figure 5.7: An overview of the alignment process. The dotted line represents the optional bootstrapping.

5.3 Phoneme recognition using models trained on a-capella singing

In this section, the most salient tested approach for phoneme recognition in singing is presented. For this method, acoustic models were trained on actual singing recordings. A large set of such recordings was already available from the *DAMP* corpus, but no annotations were included with them. Such annotations were generated automatically from text lyrics available on the internet. In the following subsections, this process is described in detail, some variants are tested, and experiments with these new models are presented.

5.3.1 Corpus construction

As mentioned above, no lyrics annotations are available for the *DAMP* data set, but the textual lyrics can be obtained from the *Smule Sing!* website[2]. All of them are English-language songs. These lyrics are mapped to their phonetic content using the CMU Pronouncing Dictionary[3] with some manual additions of unusual words. As with the other data sets, this dictionary has a phoneme set of 39 phonemes (also see appendix A.1).

These lyrics are automatically aligned to the *DAMP* audio using the HMM acoustic models trained on *TIMIT* (see section 5.1). Viterbi alignment is performed on the word and phoneme levels using the HTK framework with MFCCs and their deltas and double-deltas as features. This is the same principle of so-called "Forced Alignment" that is commonly used in Automatic Speech Recognition [156] (although it is usually

[2] `http://www.smule.com/songs`
[3] `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`

done on shorter utterances).

Several different alignment strategies were carried out:

One state vs. three states per phoneme Versions with one state per phoneme and three states per phoneme (so-called "senones", modeling the start, middle, and end phases) were tested. Since *TIMIT* only contains single-phoneme annotations, this was done by first splitting the phoneme time frames evenly in three, and then re-training the *TIMIT* acoustic models and re-aligning the data set (with the assumption that the transitions between the three states would be "pulled" to the correct times).

One-pass alignment vs. Bootstrapping On top of a one-pass alignment using the Viterbi algorithm, bootstrapping the acoustic models to improve the alignment was also tested. To clarify: The alignment is first performed on the *DAMP* data sets using the *TIMIT* models described above, and then acoustic models are trained on the resulting phoneme annotations. Then, those models are used to re-align the *DAMP* data, which is again used to train another model. This is done over three iterations.

A modified version of the alignment algorithm is used for performing the alignment with the models trained on the *DAMP* data sets. This approach is based on a DTW on the generated phoneme posteriorgrams with no punishment for very long states. This is similar to the approach described in section 8.2.

A graphical overview of the alignment process is given in figure 5.7.

Of course, errors cannot be avoided when performing automatic forced alignment. All in all, there are four combinations of these strategies, which were compared. The next section describes how this alignment procedure was validated and what strategies performed best.

Since there is usually a large number of recordings of the same song, an approach using this information to improve the alignment results was also considered, e.g. by averaging time stamps over the alignments of several recordings. This was not done in this work because recordings tend to have different offsets from the beginning (i.e. silence in the beginning), and the singers also do not necessarily pronounce phonemes at the same time. This might be an avenue for future research, though.

5.3.2 Alignment validation

The alignment approach that was used to create the new *DAMP*-based data sets was first tested on the *ACAP* data set. To recap: This approach employs models trained on

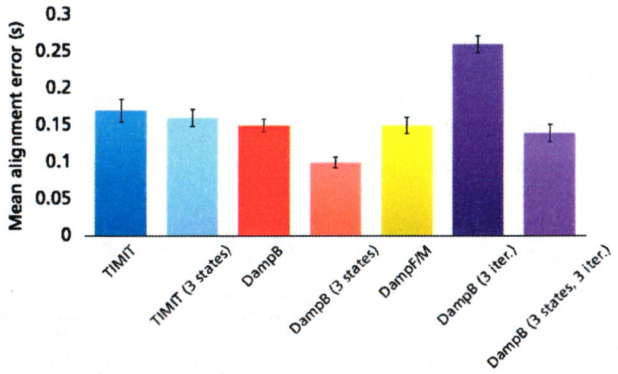
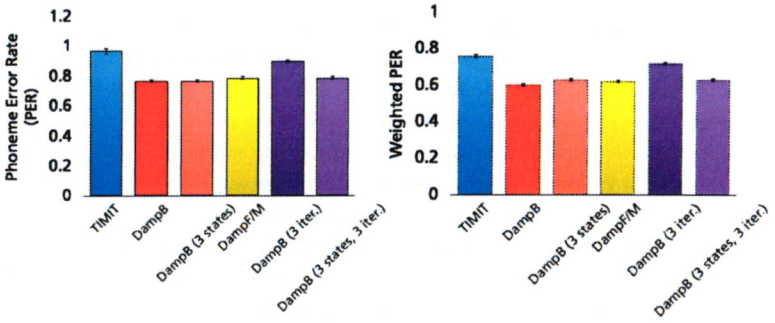


Figure 5.8: Mean alignment error in seconds on the *ACAP* data set. *TIMIT* shows the result for the same models used for aligning the new *DAMP*-based data sets. (Error bars represent standard error over the tested songs).

the *TIMIT* speech corpus, which are used for Viterbi alignment of the known phonemes to the singing. The result of this process is then compared to the manual annotations by calculating the difference between each expected and predicted phoneme transition. As mentioned in section 5.1 and shown in figure 5.8, the mean alignment error for this first approach is 0.16 seconds for the three-state case, and 0.17 seconds for single phoneme states.

Various models trained on the new *DampB*, *DampF*, and *DampM* training data sets were then tested for the same task. The results are also shown in figure 5.8.
Models trained on the monophonic alignments of *DampB* perform slightly (but significantly) better at alignment on singing data with a mean error of 0.15 seconds. For the gender-specific models, the error was only calculated for the songs of the matching gender in *ACAP*, resulting in the same average value. Since the gender-specific models do not produce better results, the other experiments were conducted with the mixed-gender training set only. The three-state version of the *DampB* model performs even better at alignment, with a mean error of 0.1 seconds. This might happen because dedicatedly training the model for the start and end parts of phonemes makes the alignment approach more accurate at finding start and end points.
Finally, a model trained on *DampB* over three iterations was also tested for align-



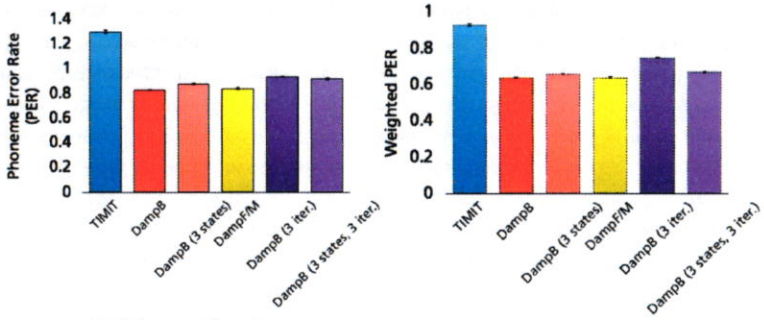
(a) Phoneme Error Rate (b) Weighted Phoneme Error Rate

Figure 5.9: Mean phoneme recognition results on the *ACAP* data set using acoustic models trained on *TIMIT* and the new *DAMP*-based data sets (error bars represent standard error over utterances).

ment. This model performs much worse at this task with a mean alignment error of 0.26 seconds. This could happen because errors in the original alignment of the phonemes become amplified over these iterations. The effect is not as pronounced for the three-state version, but it is still present.

5.3.3 Phoneme recognition

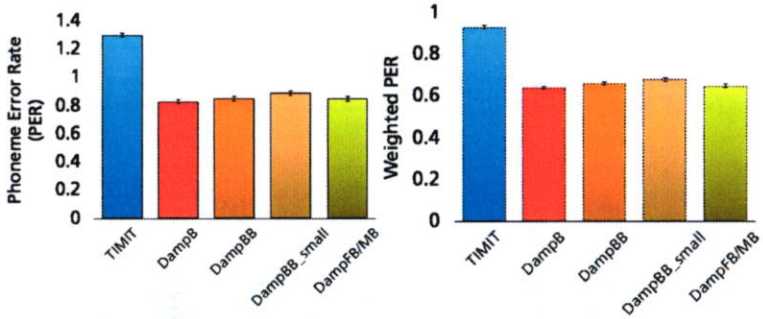
After validating the alignment strategy, phoneme recognition experiments were carried out on both the *ACAP* and the *DampTest* data sets. This was possible even though there are no manual annotations for the *DampTest* sets because the expected phonemes are available from the textual lyrics. Again, the Phoneme Error Rate and the Weighted Phoneme Error Rate are used as evaluation measures (see above).
The results for *ACAP* are shown in figure 5.9. In general, models trained on *DampB* performed much better at phoneme recognition than those trained on *TIMIT*. Compared to these speech-based models, the Phoneme Error Rate falls from 1.06 to 0.77, while the Weighted Phoneme Error Rate falls from 0.8 to 0.59. As can be seen from both evaluation measures, using alignments with three states per phoneme instead of a single state does not improve the results in this case, contrasting with the better alignment results. This might happen because more classes cause more confusion in the model, even though the three-state results were downmapped to single phonemes for calculating the evaluation measures. Additionally, the temporal pronunciation phases



(a) Phoneme Error Rate

(b) Weighted Phoneme Error Rate

Figure 5.10: Mean phoneme recognition results on the *DampTest* data sets using acoustic models trained on *TIMIT* and the new *DAMP*-based data sets (error bars represent standard error over utterances).



(a) Phoneme Error Rate

(b) Weighted Phoneme Error Rate

Figure 5.11: Mean phoneme recognition results on the *DampTest* data sets using acoustic models trained on *TIMIT* and the new *DAMP*-based data sets in various sizes (error bars represent standard error over utterances).

may just be too variable in singing, as opposed to speech. As in the alignment results, using gender-specific models does not provide an advantage over mixed-gender models. (Results for the gender-specific models were only evaluated on songs of the matching gender).
As already seen in the alignment validation results, training models on *DampB* over three iterations actually degrades the result. Again, this might happen because phoneme alignment errors are amplified over the iterations. Interestingly, the effect is not as strong for the three-state models, perhaps because the three classes per phoneme help to alleviate each other's errors.

The results of the same procedure on the *DampTest* sets are shown in figure 5.10. Results over *DampTestF* and *DampTestM* are averaged. The same general trend can be observed for these results: The Phoneme Error Rate falls from 1.3 to 0.83 when compared to models trained on *TIMIT*, with the Weighted Phoneme Error Rate decreasing from 0.93 to 0.64. Using three states per phoneme does not contribute to the result, and neither does the three-iteration bootstrapping process for training acoustic models. As in the *ACAP* results, not even the gender-specific models improve the result. This effect might occur because the range of pitch and expressions is much wider in singing than in speech, and therefore gender-specific models may not actually learn as much added helpful information. Addtional experiments indicate that gender-specific models also do not improve the results when using three states or three alignment iterations as with the mixed-gender training data.

Going forward, single-state models trained on mixed-gender data (i.e. *DampB*) appear to be the best and simplest solution. To gain insight into the role of the composition of the training data set, more experiments were conducted with the variants of *DampB* described in section 4.2.3. The results are shown in figure 5.11.
When using the smaller, phonetically balanced version of *DampB* (*DampBB*), the results become somewhat (but significantly) worse, with a Phoneme Error Rate of 0.85 and a Weighted Phoneme Error Rate of 0.66. This is particularly interesting because this data set is only 4% the size of the bigger one and training is therefore much faster. With the smallest data set which is only half the size of *DampBB*, the change is similar: The Phoneme Error Rate rises to 0.89, the Weighted Phoneme Error Rate to 0.68. Since this data set has a similar amount of phoneme instances as *TIMIT*, this proves that the improvement is actually caused by the acoustic properties of the training data, rather than just the larger amount of data. (Of course, the size is also

a contributing factor). The reduced versions of *DampF* and *DampM* were tested as well, but once again, they do not provide an improvement over the mixed-gender model.

5.3.4 Error sources

When using an automatic alignment algorithm for generating new annotations, errors cannot be avoided. To acquire a clearer picture of the reasons for the various misalignments, the audio data where they occurred was analyzed more closely. Some sources of error repeatedly stuck out:

Unclear enunciation Some singers pronounced words very unclearly, often focusing more on musical performance than on the lyrics.

Accents Some singers sung with an accent, either their natural one or imitating the one used by the original singer of the song.

Young children's voices Some recordings were performed by young children.

Background music Some singers had the original song with the original singing running in the background.

Speaking in breaks Some singers spoke in the musical breaks.

Problems in audio quality Some recordings had qualitative problems, especially loudness clipping.

For most of these issues, more robust phoneme recognizers would be helpful. For others, the algorithm could be adapted to be robust to extraneous recognized phonemes (particularly for the speaking problem). A thorough manual check of the data would be very helpful as well.

5.4 Conclusion

In this chapter, new approaches for phoneme recognition in singing were presented. As a starting point, DNN models are trained on the *TIMIT* speech corpus. For verification, these models were evaluated on the test section of *TIMIT*, resulting in a Phoneme Error Rate of 0.4 and a Weighted Phoneme Error Rate of 0.3. The results on the *ACAP* singing data set are much worse: A Phoneme Error Rate of 0.97 and a

Weighted Phoneme Error Rate of 0.76, demonstrating the difficulty of phoneme recognition in singing as opposed to speech. The Phoneme Error Rate is even worse on the *DampTest* data sets at 1.28, with a Weighted Phoneme Error Rate of 0.9. Generally, results on the *DampTest* sets are worse than on the *ACAP* test set, which presumably happens because the *DampTest* sets are performed by amateurs, whose enunciation is not as clear as that of the professional singers in the *ACAP* set and who may not always sing the correct lyrics. Additionally, the recording quality varies much more, and the annotations contain errors due to the automatic phoneme alignment (as opposed to the more reliable manual annotations of *ACAP*).

In order to better adapt the models to singing, acoustic modifications are performed on the *TIMIT* training data - namely, pitch shifting and time stretching. This increases the Phoneme Error Rate on speech test data (the *TIMIT* test section), but improves results on sung data. Pitch shifting has a stronger effect than time stretching, which might happen because this modification results in actual changed feature data, whereas time stretching mainly produces more frames with feature values similar to or in between the existing ones. On the *ACAP* test data, the lowest Phoneme Error Rate is 0.95, and the lowest Weighted Phoneme Error Rate is 0.7 (with the pitch-shifted models). On *DampTest*, those values are decreased to 0.98 and 0.76 respectively.

The best-adapted models for recognizing phonemes in singing should be those trained on actual singing data. Unfortunately, there are no big annotated singing data sets like those for speech. For this reason, the *DAMP* data set, which contains thousands of recordings of unaccompanied amateur singing, was selected, and the text lyrics were obtained from the internet. Then, various strategies for aligning the phonetic content of these lyrics to the audio were tested. In the simplest algorithm, HMM models trained on *TIMIT* are used for Viterbi alignment. This also turned out to be one of the most effective algorithms for the alignment process necessary for constructing this new singing data set.

The resulting annotations, together with the singing audio data, are then used to train new DNN models for phoneme recognition. Using more than 6,000 of these recordings of singers of both genders, the Phoneme Error Rate is lowered to 0.77 on the *ACAP* test set, and the Weighted Phoneme Error Rate is lowered to 0.59. On the *DampTest* sets, those values decrease to 0.83 and 0.64 respectively. Neither employing three states per phoneme instead of a single state nor training gender-specific mod-

els provide additional advantages. Iterating the alignment process worsens the result, possibly because errors in the original alignment become amplified over these iterations.

The influence of training set size was also investigated on the *DampTest* data. Using a phonetically balanced subset of *DAMP* that is just 4% of the size of the originally selected training set worsens the result by only 0.02 (both in Phoneme Error Rate and Weighted Phoneme Error Rate). Using an even smaller data set that is roughly the size of *TIMIT* increases the Phoneme Error Rate only by a further 0.04 (and the Weighted Phoneme Error Rate by 0.02), thereby proving that the better recognition results are caused by the sung content rather than just the size of the training data. In future experiments, it would be interesting to analyze this dependency on training set size further to find out how much data is necessary for a salient model.

A manual error analysis was performed on cases where the phoneme recognition failed. Errors are frequently caused by linguistically or acoustically difficult segments in the test data, such as accents, children's voices, background music, or additional speaking.

Comparing these results to the state of the art is difficult because of the different testing data used. Considering raw numbers, it can be assumed that the best results are in the range of the best state of the art results, for example those by Mesaros et al. [108] (see section 3.2). In contrast to these approaches, no post-processing is employed. In the future, integrating such steps as singer adaptation and language modeling would in all probability serve to improve the approach even more.
Another option that has not been tested yet is the use of triphones (i.e. training tied models for phonemes with context dependency on their predecessors and successors). This approach is frequently used in ASR.

6 Sung Language Identification

Language identification has been extensively researched in the field of Automatic Speech Recognition since the 1980's. A number of successful algorithms has been developed over the years. An overview over the fundamental techniques is given by Zissman in [157].
Fundamentally, four properties of languages can be used to discriminate between them:

Phonetics The unique sounds that are used in a given language.

Phonotactics The probabilities of certain phonemes and phoneme sequences.

Prosody The "melody" of the spoken language.

Vocabulary The possible words made up by the phonemes and the probabilities of certain combinations of words.

Even modern systems mostly focus on phonetics and phonotactics as the distinguishing factors between languages. Vocabulary is sometimes exploited in the shape of language models.
In ASR, the standard technique for language identification is Parallel Phone Recognition followed by Language Modeling (PPRLM). In this approach, acoustic and language models are trained for each language (or, in some cases, only the language models are different and just one acoustic model is used) . Unseen examples are then run through each model or combinations thereof, and the result with the highest likelihood determines the language (e.g. [158] and [159]).

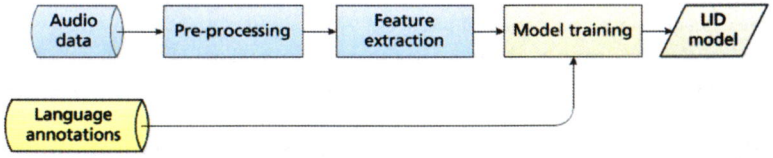


Figure 6.1: Schematic of the training procedure for language identification.

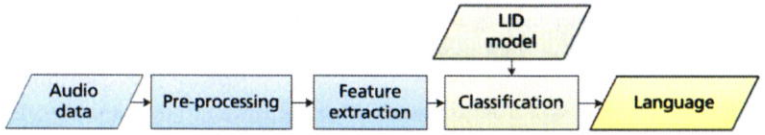


Figure 6.2: Schematic of the classification procedure for language identification.

Other approaches directly train models for each language on the feature vectors (e.g. GMMs). This technique can be considered a "bag of frames" approach, i.e. the single data frames are considered to be statistically independent of each other. The trained models then describe probability densities for certain acoustic characteristics of each language. GMM approaches used to perform worse than their PPRLM counterparts, but the development of new features has made the difference negligible [160]. They are, in general, easier to implement since only audio examples and their language annotations are required. Allen et al. [16] report results of up to 76.4% accuracy for ten languages. Different backend classifiers, such as Multi-Layer Perceptrons (MLPs) and Support Vector Machines (SVMs) [15], have also been used successfully instead of GMMs.
In this work, an approach that trains directly on the acoustic characteristics using i-vector extraction and SVMs is presented. The modifications to the general processing chain are presented in figures 6.1 and 6.2.

Additionally, a second approach based on phoneme posteriorgrams is tested. Statistics from the posteriorgrams are calculated, and then a second model is trained on these. Similar methods have been developed in ASR: Berkling presented an approach that uses sequences of recognized phonemes to discriminate between two languages (English and German), either with statistical modeling or with Neural Networks [161]. Mean errors of 0.12 and 0.07 on unseen data are achieved for the statistical approach and the Neural Network approach respectively when enough training data is available. Li, Ma, and Lee present a system where acoustic inputs are tokenized into acoustic words, which do not necessarily correspond to phonetic n-grams. Then, language classifiers are trained on statistics of the acoustic words [162]. They obtain an equal error rate of 0.05 for six languages using a universal phoneme recognizer for tokenization and SVMs for backend language recognition. Peche et al. [163] attempt a similar approach on languages with limited resources. The performance remains good even when only acoustic models trained on different languages are used.

In all of these approaches, tokenization of some sort is performed using the acoustic models. Since phoneme recognition on singing is still relatively unreliable, statistics are calculated directly on the phoneme posteriors in this work.

For evaluation, all test examples are classified into exactly one language class. Then, the accuracy (i.e. the average retrieval) is calculated:

$$Accuracy = \frac{TP}{N} \tag{6.1}$$

where TP are the True Positives, and N is the number of all documents. In ASR, the average cost measure as recommended in [164] is also used widely now; however, to remain in line with other sung language identification approaches such as those described in chapter 3, the accuracy was still used in this work.

In both approaches, the *NIST2003LRE* and *OGIMultilang* corpora were used for testing the algorithms on speech, and the *YTAcap* data set was used for singing (see chapter 4). All results are obtained using 5-fold cross-validation - i.e., models are trained on 4/5 of each data set, then the remaining 1/5 is classified with the model. This is done 5 times until each utterance has been classified. This was necessary because the data sets are relatively small, and separating them into training and test sets would not have provided enough results for a meaningful evaluation.

6.1 Sung language identification using i-vectors

As described in section 2.4.4, i-vector extraction is a feature dimension reduction technique that was originally developed for speaker recognition, but has since then been employed successfully for other tasks, including language identification. After the publication of this approach for i-vector extraction for sung language identification, it also started being used for other MIR tasks, such as artist recognition and similarity calculation [165][166].

6.1.1 Proposed system

Figure 6.3 shows a rough overview over the i-vector classification system.

A number of features were extracted from each audio file. Table 6.1 shows an overview over the various configurations used in training.

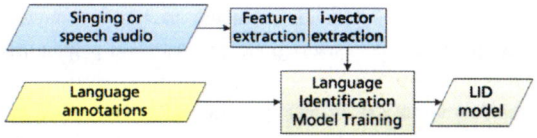


Figure 6.3: Overview of the process for language identification using i-vector extraction.

Name	Description	Dimensions
MFCC	MFCC, 20 coefficients	20
MFCCDELTA	MFCC, 20 coefficients, deltas and double-deltas	60
MFCCDELTASDC	MFCCDELTA+SDC	117
SDC	SDC with configuration 7 – 1 – 3 – 7	91
RASTA-PLP	PLP with RASTA processing, model order 13 with deltas and double-deltas	39
RASTA-PLP36	PLP with RASTA processing, model order 36 with deltas and double-deltas	96
PLP	PLP without RASTA processing, model order 13 deltas and double-deltas	39
COMB	PLP+MFCCDELTA	99

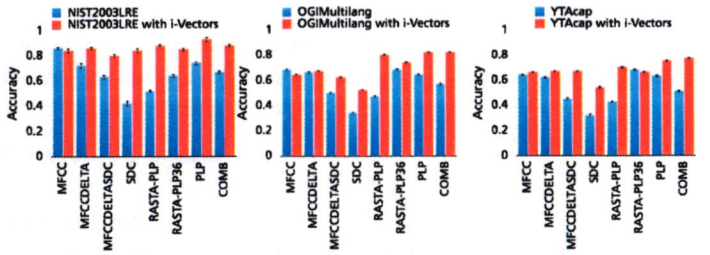
Table 6.1: Feature configurations used in language identification training.

For classification, Support Vector Machines (SVMs) are tested. The SVM parameters are determined using a grid-search. For each of the data sets, all feature combinations listed in table 6.1 are tested directly and with i-vector processing.

6.1.2 Experiments with known speakers

In the first experiment, SVM models are trained on randomly selected folds of the training data sets. This means that recordings by the same speaker are spread out between the training and test data sets. In theory, i-vectors could be particularly susceptible to capturing speaker characteristics instead of language characteristics, leading to evaluation results that are not representative of results for unknown speakers. However, this effect is often ignored in literature [167], and an argument can be made that partially training models on speaker properties is realistic for some use cases (i.e. when many of the expected speakers for each language are already known).

Figure 6.4 shows the results for the SVM models trained on the *NIST2003LRE*, *OGIMultilang*, and *YTAcap* data sets. In general, these models are able to capture the language boundaries well.
SVMs produce good results on the *NIST2003LRE* data set for all of the features. They are able to discriminate very well on this small, clean data set. The best result



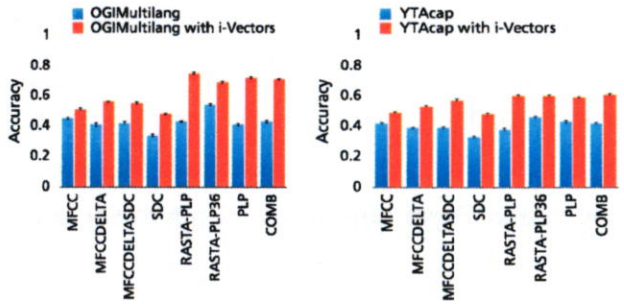
(a) NIST2003LRE (b) OGIMultilang (c) YTAcap

Figure 6.4: Results using SVM models on all three language identification data sets, with or without i-vector processing, with speakers shared between training and test sets (error bars represent standard error over training folds).

with i-vector processing is 86% accuracy for MFCC features. When using i-vectors, a 93% accuracy is achieved with PLP features. This may, in fact, be close to the upper bound for the classification here; further analysis shows that misclassified recordings often mainly consist of laughter or very few words.
The *OGIMultilang* corpus is roughly four times as big and more varied than the *NIST2003LRE* corpus, making it harder to classify. As shown, the high-dimensional pure features do not perform as well as on *NIST2003LRE*, with a maximum accuracy of 68% for MFCCs and RASTA-PLPs with 36 coefficients. Using i-vector extraction improves the result by a large margin. Feature-wise, PLPs without RASTA processing work best at a result of 82% accuracy. MFCC and SDC features did not work quite as well, but did not hurt the result either when combined with PLPs (COMB result). It is interesting to see that the i-vector extraction decreased the results for MFCCs, the feature that worked best without it.
As with all other experiments, the task becomes harder when attempted on singing data. Similar to the *OGIMultilang* corpus, the *YTAcap* corpus provides very complex and varied data. The same effects occur with the direct feature training here, too: RASTA-PLPs with 36 coefficients provide the best results, but the accuracy is not very high at 68%. i-Vector extraction once again serves to improve the result. The highest results when using i-vector extraction is a 75% accuracy when using PLP without RASTA processing, or 77% for the COMB configuration.
The same experiment was also conducted with MLP classifiers (see [168]), but they generally performed worse than the SVM classifiers. In the case of the small *NIST2003LRE* data set, strong overfitting effects were observed.

6.1.3 Experiments with unknown speakers

In order to find out what influence the speaker characteristics had on the result, the same experiments were then repeated with training and evaluation sets that strictly separated speakers. This experiment was not performed for the *NIST2003LRE* corpus because no speaker information is available for it. Since the SVM models performed better in the previous experiment, only these models were tested.



(a) OGIMultilang (b) YTAcap

Figure 6.5: Results using SVM models, with or without i-vector processing, with speakers separated between training and test sets (error bars represent standard error over training folds).

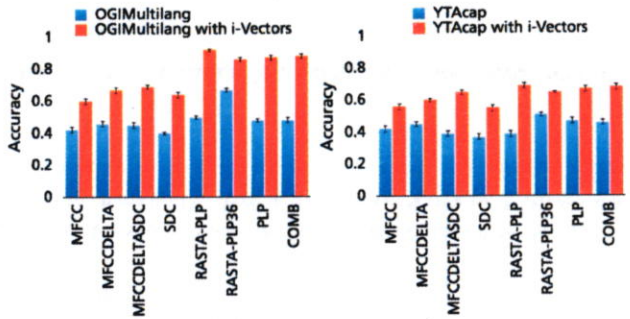
The results are shown in figure 6.5. In general, all configurations perform worse, indicating that some of the characteristics learned by the models come from the speakers rather than the languages. Apart from this, the general trends for the features remain the same, and i-vector extraction still improves the over-all results.

On the *OGIMultilang* corpus, the best result is still obtained with RASTA-PLP features and i-vector processing, but the accuracy falls by around 8 percent points to 75%. On *YTAcap*, the effect is even worse: From an accuracy of 77% with the mixed condition, the result decreases to 61% for the separated condition. The reason for this is probably the wider signal variety in singing as opposed to speech; additionally, *YTAcap* also possesses a wider range of recording conditions than the controlled telephone conditions of *OGIMultilang*. Arguably, the solution for this effect would be the use of larger training data sets, which would be able to cover these acoustic and performance conditions better. Conversely, as the previous results show, the approach produces

better results when an application scenario can be limited to a range of known speakers, or at least recording conditions (as in the *NIST2003LRE* experiment).

6.1.4 Experiments with utterances combined by speakers

All previous experiments were performed on relatively short utterances of a few seconds in duration. In many application scenarios, much more audio data is available to make a decision about the language. In particular, songs are usually a few minutes in length, and in many cases, only one result per document (= song) is required.
For this reason, results for the *YTAcap* data set are taken from the previous experiment and a majority voting decision is made for each song (and therefore also for each singer). For the *OGIMultilang* corpus, results for all utterances by the same speaker are aggregated in the same fashion, resulting in similar durations of audio. (Again, this experiment was not performed with the *NIST2003LRE* corpus due to the lack of speaker information).



(a) OGIMultilang (b) YTAcap

Figure 6.6: Document-wise results using SVM models, with or without i-vector processing (error bars represent standard error over training folds).

The results are shown in figure 6.6. Overall, aggregation of multiple utterances by the same speakers balances out some of the speaker-specific effects seen in the previous experiment. Taking more acoustic information into account, the models are able to determine the language with higher accuracy.
On the *OGIMultilang* corpus, the result is even better than on the condition with known speakers. The best result rises from 75% accuracy for short utterances to 92%

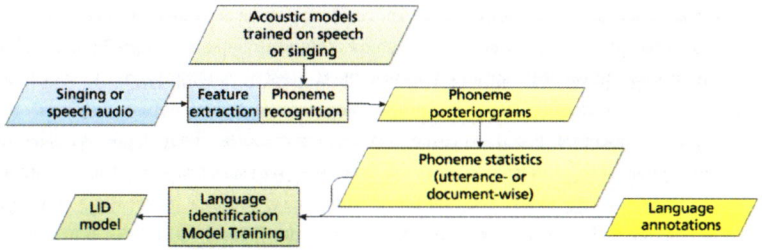


Figure 6.7: Overview of the process for language identification using phoneme statistics.

for the aggregated documents (both with the RASTA-PLP feature). On the *YTAcap* data sets, the aggregated result is 69% (compared to 60% for line segments).

As suggested in the previous section, the approach produces results that are usable in practice when the problem can be narrowed down, e.g. to known speakers or recording conditions. As this experiment shows, useful results can also be obtained when longer sequences are available for analysis.

6.2 Sung language identification using phoneme recognition posteriors

Another developed approach is based upon phoneme statistics derived from phoneme posteriorgrams. To obtain representative statistics for model training, relatively long observations are necessary, but, as described in the previous section, this is the case for many applications, for example when considering song material (e.g. songs of 3-4 minutes in duration). On the other hand, phoneme posteriorgrams need to be calculated for a number of other tasks, such as keyword spotting or lyrics-to-audio alignment.

An overview of the approach is shown in figure 6.7. Posteriorgrams are generated on the test data sets *YTAcap* and *OGIMultilang* using the acoustic models trained on the *TIMIT* speech data set and on the *DAMP* singing data set as described in section 5.3. To facilitate the following language identification, phoneme statistics are then calculated in two different ways:

Document-wise statistics Mean and variances of the phoneme likelihoods over whole songs or sets of utterances of a single speaker are calculated. This results in just two feature vectors per document (one for the means, one for the variances).

Utterance-wise statistics Means and variances of the phoneme likelihoods over each utterance are calculated (or, in the case of *YTAcap*, over each song segment). For further training, the resulting vectors for each speaker/song (= document) are used as a combined feature matrix. As a result, no overlap of speakers/songs is possible between the training and test sets.

Naturally, relatively long recordings are necessary to produce salient statistics. For this reason, the aggregation by speaker/song is done in both cases rather than treating each utterance separately.

Then, Support Vector Machine (SVM) models are trained on the calculated statistics in both variants with the three languages as annotations. Unknown song/speaker documents can then be subjected to the whole process and classified by language.

6.2.1 Language identification using document-wise phoneme statistics

In the first experiment, SVM classifiers are trained on the document-wise phoneme statistics, and classification is also performed on a document-wise basis (i.e., only one mean and one variance vector per document). The results are shown in figure 6.8.
On the singing test set, results are worst when using acoustic models trained on *TIMIT* at just 53% accuracy, and become better when using the model trained on the "songified" *TIMIT* variant *TimitM* (see section 5.2), or on the small selection of the singing training set *DampBB_small* at an accuracy of 59% each. The best result of 63% accuracy is achieved when the models are trained on the full singing data set.
Surprisingly, the results on the *OGIMultilang* corpus also improve from 75% with the *TIMIT* models to 84% using the *DampB* models. Since *TIMIT* is a very "clean" data set, training on the singing corpus provides some more phonetic variety, acting as a sort of data augmentation. This is especially important in this context where phonemes are recognized in three different languages.
On both corpora, there is no noticeable bias of the confusion matrix - i.e., the confusions are spread out evenly. This is particularly interesting when considering that the acoustic models were trained on English speech or singing only.

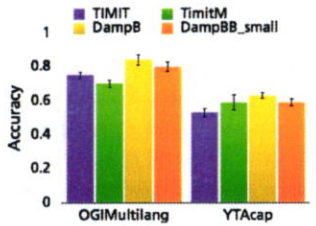


Figure 6.8: Results using document-wise phoneme statistics generated with various acoustic models (error bars represent standard error over training folds).

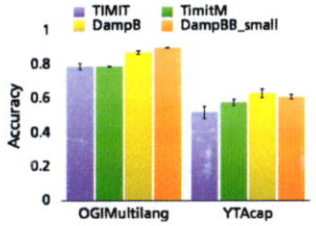


Figure 6.9: Results using utterance-wise phoneme statistics generated with various acoustic models.

6.2.2 Language identification using utterance-wise phoneme statistics

Next, language identification was performed with models trained on the statistics of each utterance contained in the document. The recognition process is still performed on the whole document. The results are reported in figure 6.9.
Phoneme statistics are not as representative when computed on shorter inputs, but they provide more information for the backend model training when utilized as a combined feature matrix for a longer document. The results on singing improve slightly (significantly) to 63% accuracy with the acoustic model trained on the small singing corpus (*DampBB_small*) and decrease insignificantly for the *DampB* model (61%). However, on the speech corpus, the best result rises to 90%.

allgemeine Frage
significantly wie ausgerechnet?

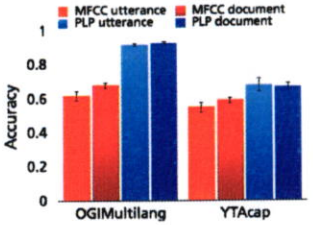


Figure 6.10: Results using utterance- and document-wise i-vectors calculated on PLP and MFCC features (error bars represent standard error over training folds).

6.2.3 For comparison: Results for the i-vector approach

For comparison, models from the previous approach were also trained on the same time scales. i-Vectors were calculated on the utterance- or the document-wise scale. This was done for PLP and MFCC features. The resulting i-vectors were then used to train SVMs in the same manner as in the previous experiments. (The difference here is that the models are already trained on the aggregated i-vectors, either with those for a whole document or with all i-vectors of the utterances constituting each document aggregated). The results are shown in figure 6.10.
The best result obtained on *YTAcap* data set is 68% accuracy. This is only 5 percent points higher than the approach based on phoneme statistics, which is easier to implement. On the *OGIMultilang* corpus, the difference is only 3 percent points (93%). Of course, the advantage of the i-vector approach is that it can also be performed on much shorter inputs.

6.3 Conclusion

In this section, two approaches to singing language identification were presented: One based on i-vector processing of audio features, and one based on the computation of phoneme statistics from posteriorgrams. In both cases, machine learning models were trained on the resulting data.

In the first approach, PLP, MFCC, and SDC features are extracted from audio data, and then run through an i-vector extractor. The generated i-vectors are then used as

inputs for SVM training. The basic idea behind the i-vector approach is the removal of language-independent components of the signal. This effectively reduces irrelevance to the language identification tasks and also reduces the amount of training data massively.
The smallest data set is the *NIST2003LRE* corpus. The SVM backend, produces good results of up to 93% for PLP features with i-vector extraction.
The *OGIMultilang* corpus is a much bigger speech corpus. Training without i-vector extraction does not work well for any feature configuration. The best accuracy for this scenario was 68%. Results of up to 83% are achieved with i-vector processing.
Language identification for singing was expected to be a harder task than for speech due to the factors described in section 3.1. The results on the *YTAcap* corpus turn out to be somewhat worse than those for the *OGIMultilang* corpus, which is of similar size. Once again, i-vector extraction improves the results from 63% to 73%.
The same experiment is repeated with no speaker overlap between training and test sets. The results fall significantly, indicating speaker influence on the model training.
In a third experiment, the results are aggregated into documents by each speaker, which again leads to improved results. The best accuracy on *OGIMultilang* is 92%, while on *YTAcap*, it is 69%. Both experiments demonstrate that useful results can be obtained when limiting the task, e.g. by training on a set of known speakers or recording conditions, or by analyzing documents of longer durations. Alternatively, a wider range of speakers in the training data would lead to models that generalize better.
Overall, i-vector extraction reduces irrelevance in the training data and thereby leads to a more effective training. As additional benefits, the training process itself is much faster and less memory is used due to its data reduction properties. Most of the state-of-the-art approaches are based on PPRLM, which requires phoneme-wise annotations and a highly complex recognition system, using both acoustic and language models. In this respect, this system is easier to implement and merely requires language annotations.

The second presented method is a completely new language identification approach for singing. It is based on the output of various acoustic models, from which statistics are generated and SVM models are trained. In contrast to similar approaches for speech, no voice tokenization is performed. Since phoneme recognition on singing is not always reliable, the statistics are calculated directly on the phoneme posteriorgrams, although this does not take any temporal information into account. The acoustic

models are trained only on English-language material (speech and singing); it would be interesting to test this with multi-language training data. Due to the statistics-based nature of the approach, it is not suited for language identification of very short audio recordings.
The accuracy of the result for singing is somewhat worse than the results obtained with the i-vector based approach. However, this new approach is much easier to implement and the feature vectors are shorter. For many applications, such posteriors need to be extracted anyway and can efficiently be used for language identification when long observations are available. The best accuracy of 63% is obtained with acoustic models trained on the *DampB* singing corpus.
Interestingly, the best result on the *OGIMultilang* speech corpus is also obtained with these acoustic models (and is only 3 percent points below the one obtained with the i-vector approach). This possibly happens because the singing corpora provide a wider range of phoneme articulations. It would be interesting to try out these acoustic models for other phoneme recognition tasks on speech where robustness to varied pronunciations is a concern.

7 Sung Keyword Spotting

In [169], three basic principles for keyword spotting in speech are mentioned:

LVCSR-based keyword spotting In this approach, a full transcription of the audio recording is performed using Large-Vocabulary Continuous Speech Recognition (LVCSR), which can then be searched for the keyword. This is expensive to implement and offers no tolerance for transcription errors - if the keyword is not transcribed correctly, it will never be found later.

Acoustic keyword spotting Acoustic KWS algorithms only search for the keyword on the basis of its acoustic properties. This approach is easy to implement and provides some tolerance for pronunciation variations. However, it does not take any a-priori knowledge about the language into account (e.g. about plausible word or phoneme sequences).

Phonetic search keyword spotting Again, a full transcription of the audio recording is performed, but the full lattices are retained instead of just the final transcription. A phonetic search for the keyword can then be run on these lattices. This approach combines the a-priori knowledge of the LVCSR-based approach with the robustness of the acoustic approach.

As described in section 3.1, there are significant differences between speech and singing signals, which means that ASR approaches for keyword spotting cannot simply be transferred to singing. In particular, both LVCSR-based keyword spotting and Phonetic search keyword spotting depend heavily on predictable phoneme durations (within certain limits). When a certain word is pronounced, its phonemes will usually have approximately the same duration across speakers. The language model employed in both approaches will take this information into account. However, phoneme durations in singing are not as predictable in speech, as figure 3.1 demonstrates. For this reason, a simpler acoustic approach using keyword-filler HMMs is employed in this work.

Keyword-filler HMMs have been described in [170] and [171]. In general, two separate

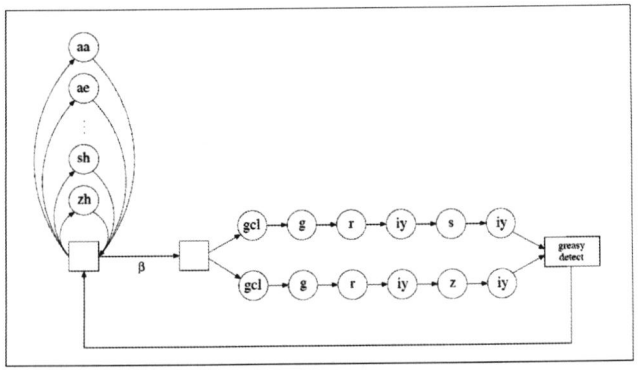


Figure 7.1: Keyword-filler HMM for the keyword "greasy" with filler path on the left hand side and two possible keyword pronunciation paths on the right hand side. The parameter β determines the transition probability between the filler HMM and the keyword HMM. [171]

HMMs are created: One for the requested keyword, and one for all non-keyword regions (=filler). The keyword HMM has a simple left-to-right topology with one state per keyword phoneme, while the filler HMM is a fully connected loop of states for all phonemes. These two HMMs are then joined. Using this composite HMM, Viterbi decoding is performed on the phoneme posteriorgrams. Whenever the Viterbi path passes through the keyword HMM, the keyword is detected. The likelihood of this path can then be compared to an alternative path through the filler HMM, resulting in a detection score. A threshold can be employed to only return highly scored occurrences. Additionally, the parameter β can be tuned to adjust the model. It determines the likelihood of transitioning from the filler HMM to the keyword HMM. The whole process is illustrated in figure 7.1.

Integration with the phoneme recognition system is shown in figure 7.2. Effectively, the keyword-filler HMM is added as a post-processing step after classification, and thus performed on the posteriorgrams.

Keyword spotting results are considered correct when they are detected within the expected songs, and are evaluated according to the F_1 measure. This measure is the harmonic mean of precision P and recall R:

$$F_1 = 2\frac{P \cdot R}{P + R} \tag{7.1}$$

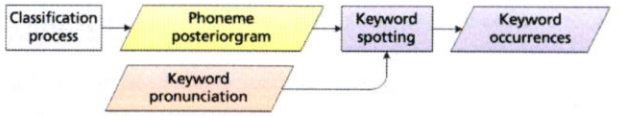


Figure 7.2: Schematic of the procedure for keyword spotting.

This measure is especially suited for cases where the classes are not balanced; in keyword spotting, occurrence of a keyword is much rarer than non-occurrence. In continuous speech recognition, the Figure Of Merit measure is often used [172]; however, since timing is not an issue in many applications for sung keyword spotting, this measure is not employed here.

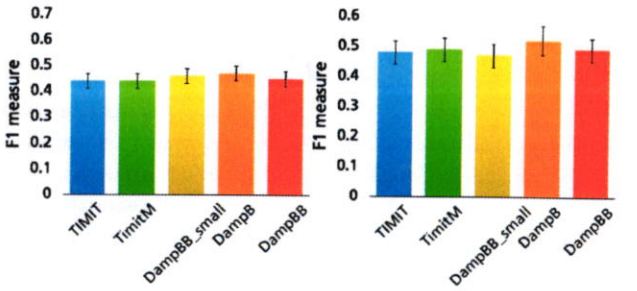
Keyword detection was performed on whole songs, which is a realistic assumption for many practical applications. The *ACAP* and *DampTest* data sets were used for evaluation with the keyword set described in section 4.4. Song-wise F_1 measures were calculated for evaluation. As in the phoneme recognition experiments, no cross validation is employed because the training and test data sets serve different purposes.

7.1 Keyword spotting using keyword-filler HMMs

7.1.1 Comparison of acoustic models

Phoneme posteriorgrams were generated with the various acoustic models described in section 5.3. The results in terms of F_1 measure across the whole *DampTest* sets are shown in figure 7.3a. Figure 7.3b shows the results of the same experiment on the small *ACAP* data set.

Across all keywords, a document-wise F_1 measure of 0.44 is obtained using the posteriorgrams generated with the *TIMIT* model on the *DampTest* data sets. This result remains the same for the *TimitM* models trained on "songified" speech. In this experiment, using models trained on the *DAMP*-based singing data sets only improves the results insignificantly, with F_1 measures of 0.47 for the *DampB* model, and 0.46 with the much smaller *DampBB_small* model. Surprisingly, in this case, the model trained on the medium-size balanced data set *DampBB* performs a little worse than the smallest one; however, this might just be due to some statistical fluctuation. In general, results on these test data sets are inconclusive. There are several reasons



(a) *DampTest* (b) *ACAP*

Figure 7.3: F_1 measures for keyword spotting results using posteriorgrams generated with various acoustic models (error bars represent standard error over keywords).

for this: First, the annotations were generated automatically and the keywords were picked from the aligned lyrics. The singers do not always perform or pronounce them correctly. Additionally, the keyword approach can be tuned easily for high recall; then, the precision becomes the deciding factor for F_1 calculation. Considering the size of the data set, keyword occurrences are relatively rare, which makes obtaining a high precision more difficult and blurs the F_1 measures between approaches.

On the hand-annotated *ACAP* test set, the differences are somewhat more pronounced (but possibly still not significant). The F_1 measure is 0.48 for the *TIMIT* model, and rises to 0.52 with the *DampB* model. The *TimitM* and *DampBB* models both produce F_1 measures of 0.49. The higher over-all values are caused by the more accurate annotations and by the higher-quality singing. Additionally, the data set is much smaller with fewer occurrences of each keyword, which emphasizes both positive and negative tendencies in the detection.

In general, recalls are usually close to 1, and precisions often in the range of 0.2 to 0.5 (with much lower and higher outliers). For this reason, an approach that could exploit a configuration with high recalls and then discard unlikely occurrences may offer an improvement. This idea is explored further in section 7.2.

7.1.2 Gender-specific acoustic models

Keyword spotting was also performed on the posteriorgrams generated with the gender-dependent models trained on *DampF* and *DampM* (also described in section 5.3). The results are shown in figure 7.4.

Similar to the phoneme recognition results from Experiment C, the gender-dependent models offer no improvements over the mixed-gender ones of the same size, and are in the same range as the one trained on much more data (*DampB*). The F_1 measures for the female test set are 0.48 for the *DampB* model, and 0.47 for both the *DampBB* and the *DampFB* model. For the male test set, they are 0.47 for the *DampB* model, and 0.45 for the other two.

7.1.3 Individual analysis of keyword results

Figure 7.5 shows the individual F_1 measures for each keyword using the best model (*DampB*), ordered by their occurrence in the *DampTest* sets from high to low (i.e. number of songs which include the song). There is a tendency for more frequent keywords to be detected more accurately. This happens because a high recall is often achievable, while the precision depends very much on the accuracy of the input posteriorgrams. The more frequent a keyword, the easier it also becomes to achieve a higher precision for it.

As shown in literature [173], the detection accuracy also depends on the length of the keyword: Keywords with more phonemes are usually easier to detect. This explains the relative peak for "every" in contrast to "think" or "night". Since keyword detection systems tend to perform better for longer words and most of the keywords only have 3 or 4 phonemes, the results achieved so far are especially interesting.

One potential source of confusion are sequences of phonemes that overlap with keywords, but are not included in the calculation of the precision. Identically spelled parts of words were included, but split phrases and different spellings were not (e.g. "away" as part of "castaway" would be counted, but "a way" would not be counted as "away"). This lowers the results artificially and could be an avenue for future improvement. Additionally, only one pronunciation for each keyword was provided, but there may be several possible.

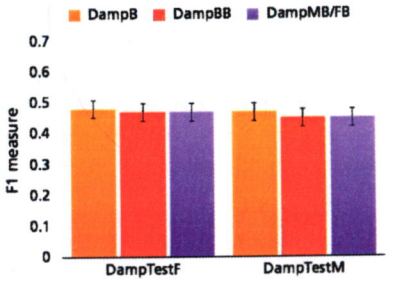


Figure 7.4: F_1 measures for keyword spotting results on the *DampTestM* and *DampTestF* data sets using mixed-gender and gender-dependent models (error bars represent standard error over keywords).

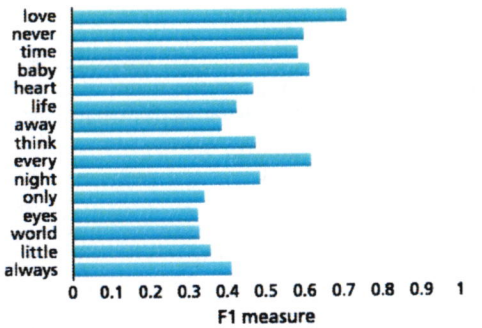


Figure 7.5: Individual F_1 measures for the results for each keyword, using the acoustic model trained on *DampB*.

7.2 Keyword spotting using duration-informed keyword-filler HMMs

7.2.1 Approach

As mentioned above, a high recall is easily achievable with the described approach, but the comparatively low precision decreases the over-all result. Therefore, using side information to reject false positives would be a helpful next step.
One such source of information are the durations of the detected phonemes. As shown in figure 3.1, each phoneme in the *TIMIT* speech database has a fairly fixed duration. In singing, the vowels' durations vary a lot, but the consonants' are still quite predictable. Standard HMMs do not impose any restrictions on the state durations, resulting in a geometric distribution which does not correspond to naturally observed distributions of phoneme durations.
As first described in [174], introducing restrictions on state durations can improve the recognition results. In [175], Juang et al. present two basic approaches for duration modeling in HMMs: Internal duration modeling and Post-processor duration modeling.

In both approaches, parametric state duration models for each phoneme need to be calculated first [176]. Several distributions has been tested for this task (e.g. Gaussian ones), but Burshtein showed that gamma distributions are best at modeling naturally occurring phoneme duration distributions [177]:

$$d(\tau) = K \exp\{-\alpha\tau\}\tau^{p-1} \tag{7.2}$$

where $\tau = 0, 1, 2, ...$ are the possible state durations in frames and K is a normalizing factor. The parameters α and p are estimated according to

$$\hat{\alpha} = \frac{E\{\tau\}}{VAR\{\tau\}}, \hat{p} = \frac{E^2\{\tau\}}{VAR\{\tau\}} \tag{7.3}$$

where E is the distribution mean and VAR is the distribution variance. An example is shown in figure 7.6. In this work, E and VAR are estimated from the phonetically annotated data sets.

In internal duration modeling, the durations are incorporated directly into the Viterbi alignment. This means that the Viterbi output will already be a state sequence that is optimal with regards to the a-priori phoneme duration knowledge. It is,

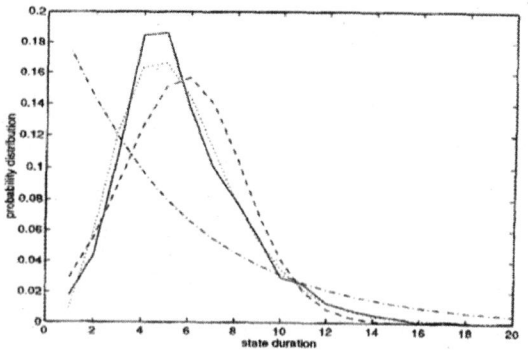


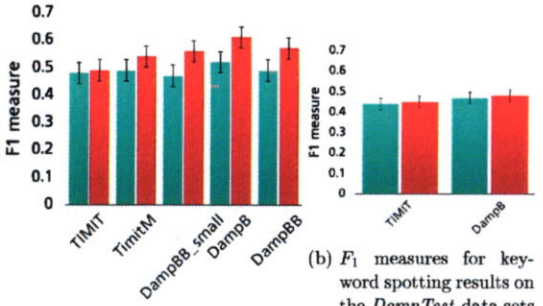
Figure 7.6: An example of the empiric duration distribution of one phoneme state (solid line) and three approximations: Gaussian (dashed), geometric(dash-dot), and gamma (dotted). [177]

however, computationally expensive. In previous experiments [178], this approach did not produce better results than the much easier to implement post-processor duration modeling. Therefore, this section will focus on that approach.

When using post-processor duration modeling, knowledge about plausible phoneme durations is imposed on the result of the Viterbi alignment, the obtained state sequence. This is computationally cheap, but only results in a new likelihood score for the calculated sequence and does not provide better possible state sequences. As described in [175], the state sequence obtained from the Viterbi alignment can be re-scored according to:

$$\log \hat{f} = \log f + \gamma \sum_{k=1}^{N} d_k(\tau_k) \tag{7.4}$$

where f is the original likelihood of the sequence, γ is a weighting factor, $k = 1...N$ are the discrete states in the state sequence, τ_k are their durations, and $d_k(\tau_k)$ is, again, the probability of state k being active for the duration τ_k.
Using keyword-filler HMMs, only one state sequence per utterance is obtained, which either contains the keyword or not. It is therefore not possible to compare these likelihood scores and equation 7.4 cannot be applied directly. To still be able to integrate post-processor duration modeling, the HMM parameters are tuned to obtain



(a) F_1 measures for keyword spotting results on the *ACAP* data set with post-processor duration modeling.

(b) F_1 measures for keyword spotting results on the *DampTest* data sets with post-processor duration modeling.

Figure 7.7: F_1 measures for keyword spotting results using posteriorgrams generated with various acoustic models with post-processor duration modeling (error bars represent standard error over keywords).

a high recall value. Then, the duration likelihood (second half of equation 7.4) is calculated for all found occurrences of the keyword and normalized by the number of states taken into account:

$$dl = \frac{1}{N} \sum_{k=1}^{N} d_k(\tau) \tag{7.5}$$

Then all occurrences where dl is below a certain threshold are discarded.

For the presented results, duration statistics from the *ACAP* data set were used, and only the consonants' durations were taken into account (since vowel durations vary much more as shown in figure 3.1). However, additional experiments showed that the result only varies slightly when using speech statistics instead, and when also discarding unlikely vowel durations. This probably happens because the keywords do not contain many states anyway, and because the duration distribution for vowels has a large variance, allowing for a wide range of durations.

7.2.2 Results

Results with and without post-processor duration modeling for the *ACAP* data set are shown in figure 7.7a. The same acoustic models as in the previous experiment were

tested. As these results show, F_1 measures improve for all configurations when post-processor duration modeling is employed. The effect is somewhat, but significantly, stronger for the *DAMP* models than for the *TIMIT* model. The best result rises from 0.52 to 0.61 with the *DampB* model. Analysis of the detailed results shows that the precision can be improved significantly when detected occurrences with implausible phoneme durations are discarded. However, this often also decreases the recall, resulting in the shown F_1 results.

The approach was also tested on the *DampTest* data sets for two acoustic models. F_1 measures on these data sets are generally blurry for the reasons described in section 7.1. In this case, the results are just a little bit higher with post-processor duration modeling.

7.3 Conclusion

In this chapter, an approach for keyword spotting using the new acoustic models trained on singing was described. Keyword spotting is performed by extracting phoneme posteriorgrams generated with these new models from the audio, and then running them through a keyword-filler HMM to detect 15 keywords. On the *DampTest* data sets, the resulting F_1 measure rises from 0.44 for the models trained on speech (*TIMIT*) to 0.47 for the new models. In general, results on the *DampTest* data sets are inconclusive because the effect of the different models is shadowed by issues with the test data itself - i.e. automatic and thus possibly inaccurate annotations, amateur singing, and a relatively low frequency of keywords because of the large size of the data sets. On the smaller, hand-annotated *ACAP* test set, the results become clearer: The best F_1 measure for the models trained on *TIMIT* is 0.48, and 0.52 with those trained on *DampB*.

This result is especially interesting because most of the keywords have few phonemes. Gender-dependent models perform similar to mixed-gender models of the same size. Individual analysis of the keyword results shows that keywords that occur more frequently are detected more accurately. This probably happens because the approach is able to obtain high recall easily, but precision is an issue. The more frequent a keyword, the easier obtaining higher precisions becomes. Additionally, keywords with more phonemes are detected more accurately than short ones because there is more information to base detection on.

One idea to improve precision was using additional information to discard implausible detections. This idea was tested in a second approach by integrating knowledge about phoneme durations. Means and variances of phoneme durations are calculated from annotated data (in particular, the *ACAP* singing data set), and then occurrences with phoneme durations outside of their gamma distributions were ignored. This approach improved F_1 measures by up to 9 percent points on the *ACAP* test data, with the best result being 0.61. On the *DampTest* data sets, the effect is still existent, but not very pronounced. This is probably because of the blurriness of the result described above.

This approach was only tested with MFCC features. As preliminary experiments suggest [93], other features like TRAP or PLP may work better on singing. So-called log-mel filterbank features have also been used successfully with DNNs [179]. Another interesting factor is the size and configuration of the classifiers, of which only one was tested (after a small grid search to validate this choice).
As in the phoneme recognition experiments, there is not much of a difference between the acoustic model trained on the more than 6,000 songs of the *DAMP* data set and the one trained on only 4% of this data. It would be interesting to find the exact point at which additional training data does not further improve the models. On the evaluation side, a keyword spotting approach that allows for pronunciation variants or sub-words may produce better results. Language modeling might also help to alleviate some of the errors made during phoneme recognition.
These models have not yet been applied to singing with background music, which would be interesting for practical applications. Since this would probably decrease the result when used on big, unlimited data sets, specialized systems would be more manageable, e.g. for specific music styles, sets of songs, keywords, or applications. Searching for whole phrases instead of short keywords could also make the results better usable in practice.
As shown in [129] and [92] and in the next chapter, alignment of textual lyrics and singing already works well. A combined approach that also employs textual information could be very practical.

8 Lyrics Retrieval and Alignment

Lyrics retrieval and alignment are related tasks: The retrieval task can be interpreted as an alignment of all possible lyrics sequences to the query audio, and a subsequent selection of the best-matching result. This is done in all described algorithms.
In this work, alignment is also performed on phoneme posteriorgrams. The lyrics with their phonetic pronunciation must be known in advance; then, an algorithm finds the best positions of each phoneme in the sequence in the posteriorgram. Traditionally, Viterbi decoding is used for this, with the transition matrix shaped such that only transitions through the expected phonemes are possible. The general process is shown in figure 8.1.
For retrieval, alignment is performed on a whole database of lyrics instead of just those for a single song as illusted in figure 8.2. Then, the scores for each alignment are compared, and the highest one determines the best match.

As a starting point, a traditional HMM-based algorithm for Forced Alignment was tested. Building on top of the posteriorgrams extracted with the new acoustic models as described in chapter 5, a new algorithm using Dynamic Time Warping (DTW) was developed and evaluated for both alignment and retrieval. Next, another processing step for explicitly extracting phonemes from the posteriorgrams was included; the resulting method was also tested for both alignment and retrieval. The general process is shown in figure 8.1.

For alignment evaluation, the approaches were submitted to the *MIREX* 2017 chal-

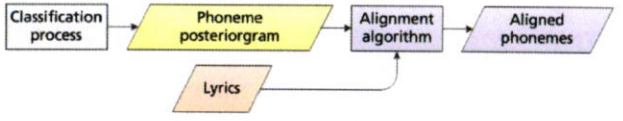


Figure 8.1: Schematic of the procedure for lyrics-to-audio alignment.

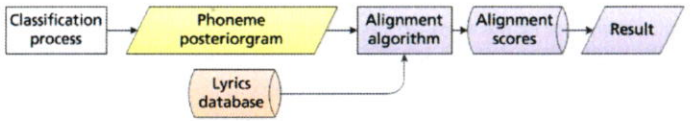


Figure 8.2: Schematic of the procedure for lyrics retrieval.

lenge for lyrics-to-audio alignment[1], in which the algorithms were tested on Hansen's dataset, both in the singing-only and polyphonic conditions (*ACAP* and *ACAP_Poly*, see sections 4.2.2 and 4.3.3), and on Mauch's alignment data set (*Mauch*, see section 4.3.2). In this challenge, the mean absolute error in seconds was used as the evaluation measure. For its calculation, all absolute deviations between the expected phoneme start and end timestamps and the automatically detected ones are calculated, averaged over the whole song, and once again over all songs. The measure was suggested in [129]. The full results can be viewed on `http://www.music-ir.org/mirex/wiki/2017:Automatic_Lyrics-to-Audio_Alignment_Results`.

und was ist das Ergebnis?

For evaluation of the retrieval results, the accuracy when taking the first N number of results into account (also called recognition rate in the state of the art) is calculated. For clarification: The first N results are checked for occurrence of the expected result; then, the percentage of queries for which the correct result was detected is calculated. (In literature, this is sometimes called the retrieval rate or recognition rate.) An alternative measure for tasks like this is the Mean Reciprocal Rank (i.e. the average inverse of the position where the expected result occurs); once again, the previously described measure was used for comparability with the state of the art. Retrieval is tested on the *AuthorRetrieval*, *DampRetrieval*, and *DampTest* data sets. Since these are dedicated test sets and model training is performed on much larger data sets, no cross validation is necessary ~~once again.~~

This chapter also presents a practical application for lyrics alignment: Automatic expletive detection in rap songs.

[1] `http://www.music-ir.org/mirex/wiki/2017:Automatic_Lyrics-to-Audio_Alignment`

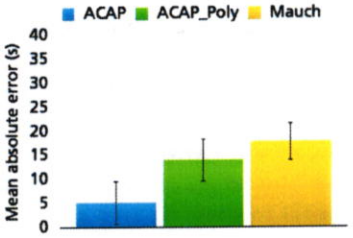


Figure 8.3: *MIREX* results on all three data sets using the HMM-based approach (error bars represent standard error over the tested songs).

8.1 HMM-based lyrics-to-audio alignment

For this algorithm, monophone HMMs are trained on the *TIMIT* speech data using the HTK framework [146]. Then, Viterbi alignment is run to align the phonemes to the singing audio. These are the same models used to align lyrics to the *DAMP* audio data to generate the training data set described in section 5.3. The results in the *MIREX* challenge are shown in figure 8.3. The detailed results for all tested songs are given in appendix A.4.

On the unaccompanied data set, the mean error is 5.11s, while the median error is 0.67s. The large difference comes about because one of the tested songs ("Clocks") produced high error values in every approach. Considering only the other songs, the highest error is 1.45s, and all others are lower than 1s. This is a good result, especially considering that the approach is relatively simple and trained on speech data only. A manual check suggests that the results are usable in practice.

On the accompanied version of the same data set, the average error is 13.96s and the median error is 7.64s. On the *Mauch* data set, the mean error is at 17.7s and the median error is at 10.14s. Once again, the error varies widely across the songs. As expected, the result is worse since no measures were taken make the approach robust to background music. Interestingly, other submitted approaches that employ source separation did not fare much better on this data set either, suggesting that the core algorithm may be better. It would be interesting to see results for the HMM-based algorithm on polyphonic music with a source separation step (or at least a vocal activity detection step) included.

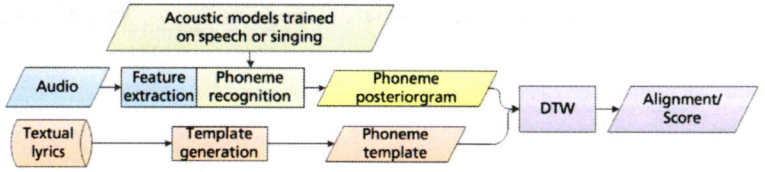


Figure 8.4: Overview of the DTW-based lyrics alignment and retrieval method.

8.2 Posteriorgram-based retrieval and alignment

A new approach is based on the posteriorgrams generated with the DNN acoustic models described in section 5.3; the procedure is shown in figure 8.4. In order to align lyrics to these posteriorgrams, binary templates are generated from the text lyrics on the phoneme scale. These can be seen as oracle posteriorgrams, but do not include any timing information.

Between this template and the query posteriogram, a similarity matrix is calculated using the cosine distance. On the resulting matrix, DTW is performed using the implementation from [30] to obtain the alignment. An example is shown in figure 8.5.

Two optimizations were made to the algorithm. The first one is a sub-sampling of the phoneme posteriorgrams by the factor 10 (specifically, the mean for 10 consecutive frames is calculated). This increases the speed of the DTW for comparisons and also produces better results. Longer windows were also tested, but this had a negative impact on the result.

Secondly, squaring the posteriorgrams before the similarity calculation produces slightly better results. This makes the posteriorgrams more similar to the binary lyrics templates. Binarizing them was also tested, but this emphasized phoneme recognition errors too much.

In the retrieval case, the binary templates are generated for all possible lyrics in the database, and the DTW calculation is performed for each of them with the query posteriorgram. The DTW cost is used as the measure to obtain the best-matching lyrics. Since the phoneme durations in the actual recording and the lyrics templates have different lengths, the length of the warping path should not be a detrimental factor in cost calculation. Therefore, the accumulative cost of the best path is divided by the path length and then retained as a score for each possible lyrics document. In the end, the lyrics document with the lowest cost is chosen as a match (or, in some

experiments, the N documents with the lowest costs).

As an additional experiment, both the textual lyrics corpus and the sung inputs were split into smaller segments roughly corresponding to one line in the lyrics each (around 12,000 lines). The retrieval process was then repeated for these inputs. This allowed an evaluation as to how well lyrics can be retrieved from just one single sung line of the song. (Sub-sequence DTW could also be used for this task instead of splitting both corpora.)

das ist eine realistische Anwendung

8.2.1 Alignment experiments

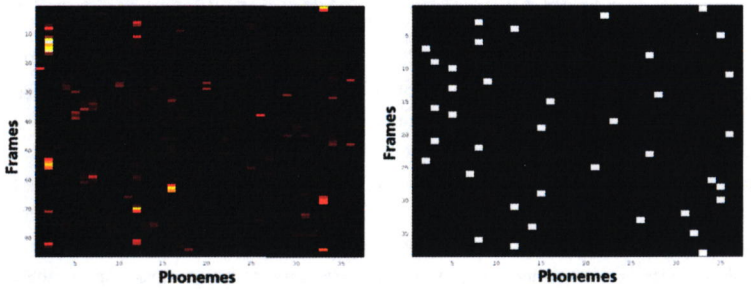
The results for this approach in the *MIREX* challenge are displayed in figure 8.6, with the detailed results given in appendix A.4. Overall, errors are much higher than with the HMM-based approach: The mean error is 9.77s for the *ACAP* data set, 27.94s on the *ACAP_Poly* data set, and 22.23s on the *Mauch* data set. This presumably happens because in contrast to the HMM approach, the algorithm does not have any information about phoneme priors and transition probabilities, neither implicitly nor explicitly. DTW on the posteriorgram is a relatively rudimentary method for performing alignments. Nevertheless, a manual check of the results suggests that the algorithm is still able to produce usable results in many cases, with many song-wise alignment errors for the *ACAP* data set still being below 1s. In particular, it works acceptably when the posteriorgram is not too noisy. This is also corroborated by the retrieval results for unaccompanied queries presented in the following.

As a future step, incorporating phoneme transition information (e.g. in the form of DNN-HMMs) could help mitigate some of the noise in the posteriorgram caused by model inaccuracies or by background music.

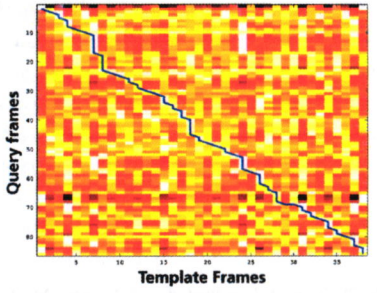
8.2.2 Retrieval experiments on whole-song inputs

In the first retrieval experiment, similarity measures were calculated between the lyrics and recordings of whole songs using the described process. This was tested with phoneme posteriorgrams obtained with all five acoustic models on the female and the male test sets (*DampTestF* and *DampTestM*). The accuracy was then calculated on the Top-1, -3, and -10 results for each song (i.e., how many lyrics are correctly detected when taking into account the 1, 3, and 10 lowest distances?). The results on the female test set are shown in figure 8.7a, the ones for the male test set in figure 8.7b.

These results show that phoneme posteriorgrams obtained with models trained on



(a) Phoneme posteriorgram (b) Phoneme template



(c) Similarity matrix with cheapest path (blue)

Figure 8.5: Example of a similarity calculation: Phoneme posteriorgrams are calculated for the audio recordings **(a)**. Phoneme templates are generated for the textual lyrics **(b)**. Then, a similarity matrix is calculated using the cosine distance between the two, and DTW is performed on it **(c)**. The accumulated cost divided by the path length is the similarity measure.

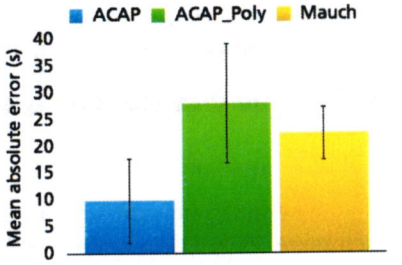
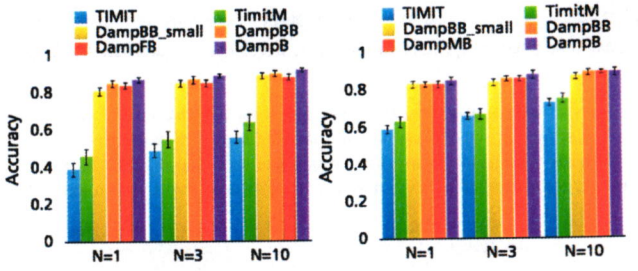


Figure 8.6: *MIREX* results on all three data sets using the DTW-based approach (error bars represent standard error over the tested songs).



(a) DampTestF (b) DampTestM

Figure 8.7: Accuracies of the results for lyrics detection on the whole song for the *DampTest* sets using the DTW-based approach with five different acoustic models, and evaluated on the Top-1, -3, and -10 results (error bars represent standard error over the tested queries).

speech data (*TIMIT*) generally produce the lowest results in lyrics retrieval. The difference between the two test sets is especially interesting here: On the male test set, the accuracy for the single best result is 58%, while on the female set it is only 39%. Previous experiments showed that the phoneme recognition itself performs significantly worse for female singing inputs, which is compounded in the lyrics retrieval results. This may happen because the frequency range of female singing is even further removed from that of speech than the frequency range of male singing is [180]. Even female speech is often performed at the lower end of the female singing frequency range. The frequency range of male singing is better covered when training models on speech recordings (especially when speech recordings of both genders are used).

This effect is still visible for the *TimitM* models, which is the variant of *TIMIT* that was artificially made more "song-like". However, the pitch range was not expanded too far in order to keep the sound natural.

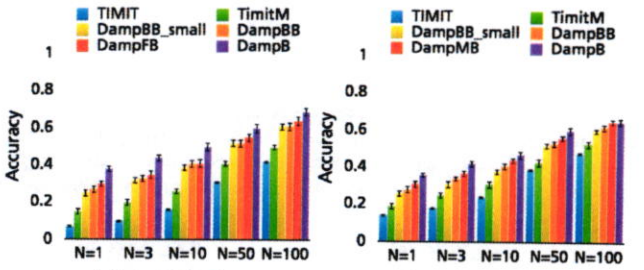
The results improve massively when acoustic models trained on any of the *DAMP* singing corpora are used. The difference between the male and female results disappears, which supports the idea that the female pitch range was not covered well by the models trained on speech. Using the models trained on the smallest singing data set (*DampBB_small*), which is slightly smaller than *TIMIT*, the results increase to 81% and 83% for the single best result on the female and the male test set respectively. With the models trained on the *DampBB* corpus, which is about twice as big, they rise slightly (but significantly) more to 85% on the female test set. Gender-specific models of the same size do not improve the results.

Finally, the results obtained with the acoustic models trained on the largest singing corpus (*DampB*) provide the very best results at accuracies of 87% and 85% (female/-male).

For some applications, working with the best N instead of just the very best result can be useful (e.g. for presenting a selection of possible lyrics to a user). When the best 3 results can be taken into account, the accuracies on the best posteriorgrams rise to 89% and 88% on the female and male test sets respectively. When the best 10 results are used, they reach 92% and 89%.

8.2.3 Lyrics retrieval experiments on line-wise inputs

In the second retrieval experiment, the same process was performed on single lines of sung lyrics as inputs (usually a few seconds in duration). Costs are calculated between the posteriorgrams of these recordings and all 12,000 available lines of lyrics. Lines with fewer than 10 phonemes are not taken into account.



(a) DampTestF (b) DampTestM

Figure 8.8: Accuracies of the results for lyrics detection on separate lines of sung lyrics for the *DampTest* sets using the DTW-based approach with five different acoustic models, and evaluated on the Top-1, -3, -10, -50, and -100 results (error bars represent standard error over the tested queries).

Then, evaluation was performed as to whether a line from the correct song was retrieved in the Top-N results. In this way, confusions between repetitions of a line in the same song do not have an impact on the result. However, repetitions of lyrical lines across multiple songs are a possible source of confusion. The results for the female test set are shown in figure 8.8a, the ones for the male test set in figure 8.8b.

Again, a difference between both test sets is visible when generating posteriograms with the *TIMIT* models. The accuracy on the best result is 14% for the male test set, but just 7% for the female test.

The results for the *DAMP* models show the same basic tendencies as before, although naturally much lower. For the single best result, the accuracies when using the *DampB* model are 38% and 36% on the female and male test sets respectively. For this task, gender-dependent models produce slightly higher results than the mixed-gender ones of the same size (possibly not statistically significant).

8.3 Phoneme-based retrieval and alignment

Next, another step was introduced to improve the previous approach and make it more flexible. This step serves to compress the posteriorgrams down to a plausible sequence of phonemes, which can then be used to search directly on a textual lyrics database. Text comparison is much cheaper than the previous comparison strategy, and enables

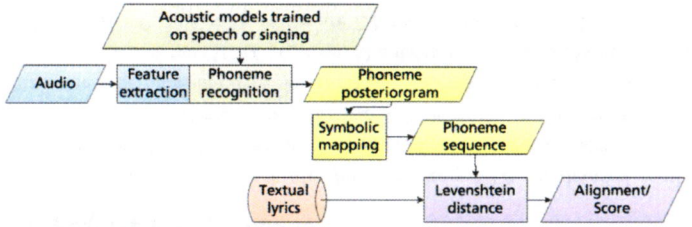


Figure 8.9: Overview of the phoneme-based lyrics retrieval process.

quick expansion of the lyrics database.
In parallel, the lyrics database is prepared by converting it into phoneme sequences as described in section 5.3.1. The key algorithms are (a) how to generate plausible phoneme sequences from the posteriorgrams, and (b) how to compare these against the sequences in the database. These parts will be described in more detail in the following. An overview is given in figure 8.9.

Symbolic mapping As described before, starting from the sung recording used as a query, MFCC features are extracted and run through the acoustic model trained on singing to recognize the contained phonemes. This produces a phoneme posteriorgram, such as the one shown in figure 5.3; i.e., probabilities of each phone over each time frame. These probabilities contain some noise, both due to inaccuracies in the model and due to ambiguities or actual noise in the performance.
(In ASR, HMMs are commonly used for obtaining phoneme sequences from the output of an acoustic model. This was not done in this work for various reasons. One of them is the lack of reliable training data; the same data as for the acoustic models could be used, but may lead to errors in the model due to the relatively small number of individual songs, and this could also amplify errors in the automatic annotation. In addition to this, the presented approach is more flexible and allows taking knowledge about phoneme performances in singing into account, such as occurrences of long phonemes and the most frequent confusions between phonemes. Nevertheless, a HMM approach could also be tested in the future).

The following steps are undertaken to obtain a plausible phoneme sequence from the posteriorgram:

1. **Smoothing** First, the posteriorgram is smoothed along the time axis with a window of length 3 in order to remove small blips in the phoneme probabilities.

2. **Maximum selection and grouping** Then, the maximum bin (i.e. phoneme) per frame is selected, and consecutive results are grouped. An example is given in figure 8.10a.

3. **Filtering by probability and duration** These results can then be pre-filtered to discard those that are too short or to improbable. This is done with different parameterizations for vowels and consonants since vowels are usually longer in duration. This yields a first sequence of phonemes, each with duration and sum probability information, which is usually too long and noisy. In particular, a lot of fluctuations between similar phonemes occur.

4. **Grouping by blocks and filtering through confusion matrix** This problem is solved by first grouping the detected phonemes into blocks, in this case vowel and consonant blocks (shown in figure 8.10b). Then, a decision needs to be made as to which elements of these blocks are the "true" phonemes and which ones are noise. This is done by taking each phoneme's probability as well as the confusion between phonemes into account. The confusion is calculated in advance by evaluating the classifier on an annotated test set; the result covers both the confusion by inaccuracies in the classifier as well as perceptual or performance-based confusions (e.g. transforming a long `/ay/` sound into `/aa/ - /ay/ - /iy/` during singing). An example of such a confusion matrix is shown in figure 8.11. The product of the probabilities and the confusions are calculated for the highest combinations up to a certain threshold, and all other detected phonemes are discarded. This results in a shorter, more plausible phoneme sequence (figure 8.10c).

Distance calculation Then, the distances between the extracted phoneme sequence and the ones provided in the lyrics database are calculated.
First, an optional step to speed up the process is introduced. Each sequence's number of vowels is counted in advance, and the same is done for the query sequence. Then, only sequences with roughly the same amount of vowels are compared (with some tolerance). This slightly decreases accuracies, but drastically speeds up the calculation time.
The similarity calculation itself is implemented with a modified Levenshtein distance.

```
['s', 'sh', 's', 'ow', 'w',
'hh', 'ah', 'sh', 's', 't', 's',
'sh', 'n', 'm', 'uw', 'b', 'ey',
'iy', 'ey', 'b', 'ih', 'iy',
'ih', 'n', 'd', 'ah', 'uw',
'ow', 'uw', 'ah', 'aa', 'ah',
'aa', 'ow', 'aa', 'uw', 'ow',
'ah', 'n', 't', 'uw', 'y', 'uw',
'n', 'uw', 'r', 'ah', 'ae',
'ay', 'ae', 'ay', 'ae', 'ay',
'er', 'ay', 'ey', 'iy', 'iy',
'iy', 'ey', 'iy', 'uw']
```

(a) Original phoneme sequence

```
['s', 'sh', 's']
[2.98, 0.32, 3.25]
['ow']
[7.13]
['w', 'hh']
[2.06, 2.50]
['ah']
[4.13]
['sh', 's', 't', 's', 'sh', 'n', 'm']
[0.21, 0.89, 0.81, 0.61, 3.77, 3.01, 0.64]
['uw']
[2.05]
['b']
[2.56]
['ey', 'iy', 'ey']
[1.81, 1.93, 1.07]
…
```

(b) Grouped into blocks (with probabilities in italics)

```
['s', 'ow', 'w', 'hh',
'ah', 's', 't', 'sh',
'n', 'uw', 'b', 'ey',
'iy', 'b', 'ih', 'n',
'd', 'uw', 't', 'uw',
'n', 'uw', 'r', 'ay']
```

(c) Filtered result

Figure 8.10: Example of the block grouping of the phoneme sequence and subsequent filtering by probabilities and confusions.

Again, the classifier's confusion between phonemes is taken into account. These confusions are used as the Levenshtein weights for substitutions. Surprisingly, using them for insertion weights improves the results as well. This probably happens because of the effect described above: A singer will in many cases vocalize an phoneme as multiple different ones, particularly for vowels with long durations. This will result in an insertion in the detected phoneme sequence, which is not necessarily a "wrong" classification by the acoustic model, but does not correspond to the expected sequence for this line of lyrics. For this reason, such insertions should not be harshly penalized. For deletions, the weight is set to 0.5 to balance out the lower insertion and substitution weights.

8.3.1 Alignment experiments

MIREX results for this approach are shown in figure 8.12; the detailed results can be found in appendix A.4. Over-all, this approach produces much lower error values than the other two, and was in fact the winning algorithm in the competition. This confirms that the phoneme detection strategy is a feasible alternative to the HMM-based approach. On the *ACAP* data set, the mean error is at 2.87s and the median is 0.26s. On *ACAP_Poly*, these values are 7.34s and 4.55s respectively, and on *Mauch*,

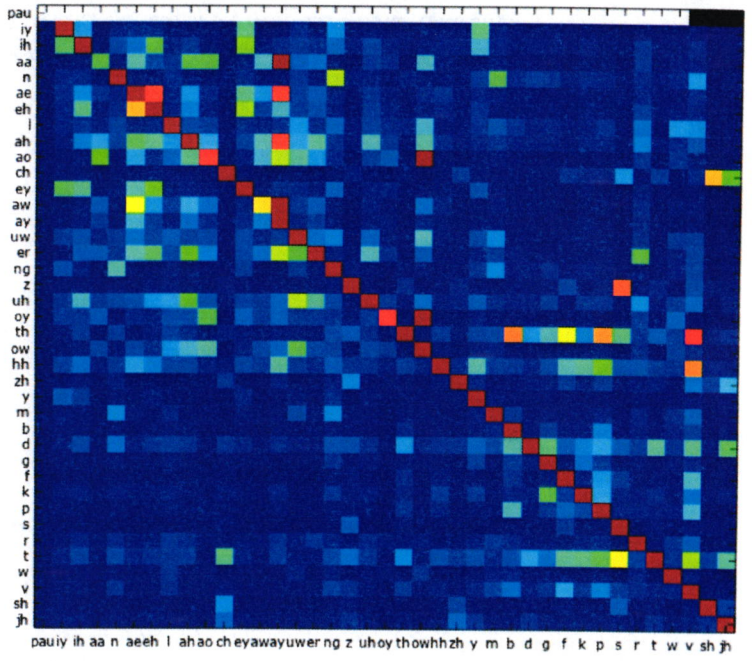


Figure 8.11: Example of a confusion matrix for an acoustic model. (Note that /pau/ confusions are set to 0).

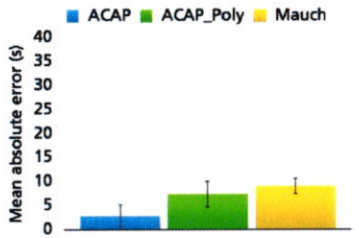


Figure 8.12: *MIREX* results on all three data sets using the phoneme-based approach (error bars represent standard error over the tested songs).

they are 9.03s and 7.52s. Once again, the results are much higher on polyphonic data than on unaccompanied singing, for which the models were trained. However, the error is still lower than with the submitted methods that include source separation. Therefore, future experiments that also employ pre-processing of this kind would be very interesting. Looking at the song-wise results, songs with heavier and noisier accompaniment generally receive higher errors than others.

8.3.2 Retrieval experiments: Calibration

For calibrating the algorithm's parameters quickly, two smaller data sets were used: *DampRetrieval*, which consists of 20 female and 20 male hand-selected sung segments with clear pronunciation and good audio quality, and *AuthorRetrieval*, which consists of 90 small performances of phrases in the *DAMP* data set by the author. Both are described in sections 4.2.3 and 4.2.4. This was done to ensure that the results were not influenced by flaws in the data set (such as unclear or erroneous pronunciation or bad quality). Lyrics can still come from the whole *DAMP* lyrics set, resulting in 12,000 lines of lyrics from 300 songs.

Figure 8.13 shows an overview over the experimental results. Accuracies are reported for the Top-1 result (i.e. the one with the lowest Levenshtein distance), the Top-3, and the Top-10 results. Queries were allowed to be one to three consecutive lines of songs, increasing the number of "virtual" database entries to around 36,000 (12,000 lines as 1-, 2-, and 3-grams). It should be noted that a result counts as correct when the correct song (out of 300) was detected; this was done as a simplification because the same line of lyrics is often repeated in songs. For possible applications, users are most probably interested in obtaining the correct full song lyrics, rather than a specific

line. Picking random results would therefore result in an accuracy of 0.3%.

The various improvement steps of the algorithm were tested as follows:

Baseline This is the most straightforward approach: Directly pick the phonemes with the highest probabilities for each frame from the posteriorgram, group them by consecutive phonemes, and use the result of that for searching the lyrics database with a standard Levenshtein implementation. This results in accuracies of 25%, 10%, and 23% for the Female, Male, and Author test sets respectively.

Posteriorgram smoothing This is the same as the baseline approach, but the posteriorgram is smoothed along the time axis as described in paragraph 8.3. This already improves the result by around 40 percent points for each test set.

Filtering blocks by probabilities and confusions This includes the last step described in paragraph 8.3. The result is improved further by 13% to 25% accuracy.

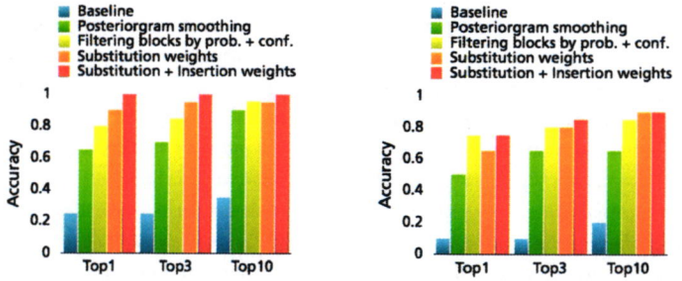
Substitution and insertion weights Finally, the modified Levenshtein distance as described in paragraph 8.3 is calculated. When using the confusion weights for phoneme substitutions only, the result increases further, and even more so when they are also used for the insertion weights. The final Top-1 accuracies are 100%, 75%, and 81% for the three test sets respectively.

8.3.3 Retrieval experiments: Full data set

The full algorithm was also tested on the full *DampTest* data sets in the same way as the DTW-based approach.

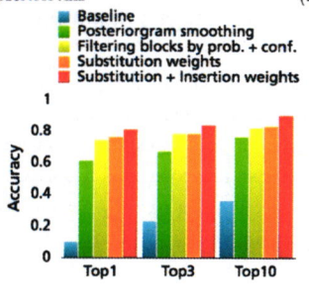
The results for retrieval using whole songs as queries are shown in figure 8.14. All the steps presented in the previous section are included in the calculation of these results. As in the DTW-based approach, the material on which the acoustic model is trained plays a huge role. When using models trained on *TIMIT*, the accuracy of the Top-1 result is just 58% on the female test set, and 75% on the male one (this mirrors the discrepancy between female and male results already seen in the previous approach). Interestingly, the model trained on *TimitM* actually performs worse for this approach with accuracies of 44% and 62% for the female and male test sets respectively. Since this approach is based on extracting salient phonemes from the posteriorgrams, it is possible that the time-stretched and pitch-shifted training data causes too much noise in the model's results, or a stronger bias towards vowels.

As previously seen, the models trained on the *DAMP* data sets perform much better



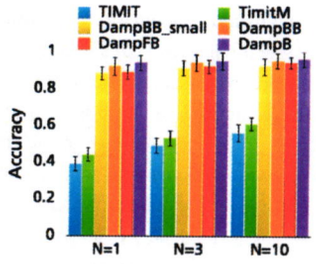
(a) DampRetrievalF

(b) DampRetrievalM

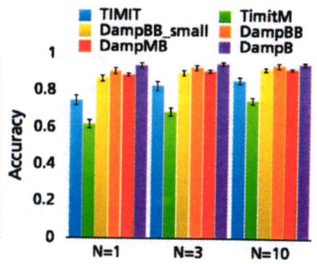


(c) AuthorRetrieval

Figure 8.13: Results of the phoneme-based retrieval algorithm with various improvement steps for three small calibration data sets.



(a) DampTestF



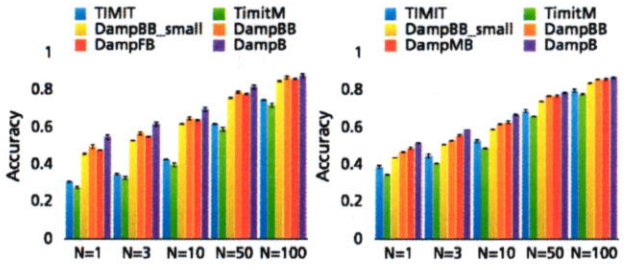
(b) DampTestM

Figure 8.14: Accuracies of the results for lyrics detection on the whole song for the *DampTest* sets using the phoneme-based approach with five different acoustic models, and evaluated on the Top-1, -3, and -10 results (error bars represent standard error over the tested queries).

at this task. On both the female and male test sets, the best result is an accuracy of 94% with the model trained on *DampB*. The models trained on less data gradually perform a few percent points worse. Interestingly, training on gender-specific data once again does not improve the result. As suggested before, this could be due to the higher variety in timbre and pitch across songs and singers compared to variation between genders.

The accuracies for the Top-3 and Top-10 results follow a similar pattern. For the *DAMP*-based models, the increase is not very high because the Top-1 result is already close to an upper bound. Analysis of the non-retrieved songs mainly shows discrepancies between the expected lyrics and the actual performance, such as those described in section 5.3.4. This means that the most effective way to improve results would lie in making the algorithm more robust to such variances (versus improving the detection itself). Over-all, the results are significantly higher than with the DTW approach.

Figure 8.15 shows the analogous results when using single line queries for retrieval. The trend across the different classifiers runs parallel to the whole song results, with the *TimitM* models performing worse than the *TIMIT* models, and both being outperformed by the *DAMP*-based models. The best Top-1 result is 55% for the female test set, and 52% for the male one. For the Top-10 results, the retrieval results are 7% and 67% respectively. Once again, lines with fewer than 10 phonemes were excluded; however, lines can still occur in more than one song. Considering this fact and the pre-



(a) DampTestF (b) DampTestM

Figure 8.15: Accuracies of the results for lyrics detection on separate lines of sung lyrics for the *DampTest* sets using the phoneme-based approach with five different acoustic models, and evaluated on the Top-1, -3, -10, -50, and -100 results (error bars represent standard error over the tested queries).

viously mentioned interfering factors, this result is already salient. When comparing these results to the ones for the calibration data set, it becomes clear that performance is highly dependent on the quality of the queries. In a practically usable system, users could be asked to enunciate clearly or to perform long segments.

8.4 Application: Expletive detection

Lots of song lyrics contain expletives. There are many scenarios in which it is necessary to know when these words occur, e.g. for airplay and for the protection of minors. During broadcasting, they are commonly "bleeped" or acoustically removed. The alignment strategies described previously are employed for the practical scenario of finding such expletives automatically.
The test data set is the one compiled by Queen Mary University, described in section 4.3.1.

A direct keyword spotting approach was also considered, but this did not generate sufficient results since most of the expletives only consist of 2 or 3 phonemes. Keyword spotting becomes notoriously hard for such short keywords as described in chapter 7. Since textual lyrics are usually easily available on the internet, a new approach utilizing those was developed:

1. Automatically align textual lyrics to audio (as described above)

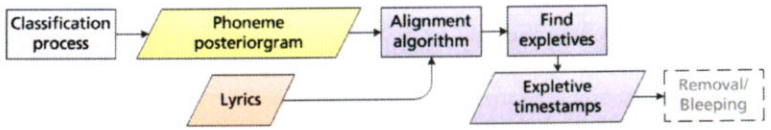
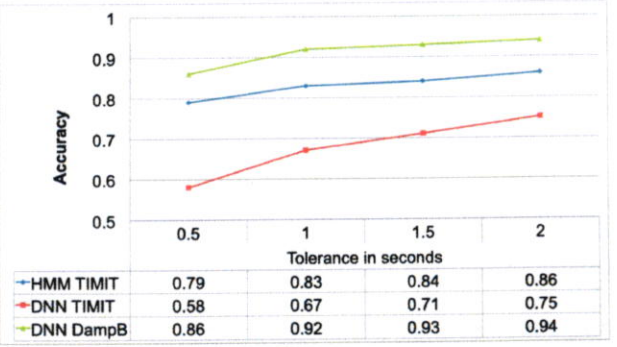


Figure 8.16: Data flow in the expletive detection approach.



	0.5	1	1.5	2
HMM TIMIT	0.79	0.83	0.84	0.86
DNN TIMIT	0.58	0.67	0.71	0.75
DNN DampB	0.86	0.92	0.93	0.94

Figure 8.17: Results for the expletive detection approach at various tolerances.

2. Search for pre-defined expletives in the result
3. If necessary, remove those expletives. A stereo subtraction approach was used, which works adequately for this case since the removed timespans are short. Alternatively, keywords can be masked with a bleep or similar.

The data flow is shown in figure 8.16.

The alignment is performed using the HMM-based alignment described in section 8.1, and the DTW alignment from section 8.2 using the DNN acoustic models trained on *TIMIT* and on *DampB*. Accuracies are then calculated by evaluating how many of the annotated expletives were recognized at their correct timestamps (with various tolerances). The results are shown in figure 8.17.

In this small practical example, only two alignment strategies were tested, but there are others that could provide better results. Whenever the alignment failed, it was mostly due to solo instruments. In order to remedy this, vocal detection (and possibly source separation) could be employed prior to alignment. Additionally, a more sophisticated removal approach could be implemented to remove the expletives.

Lyrics collected from the internet are often incorrect, e.g. because repetitions are not spelled out, because of spelling errors, or because they refer to different versions of a song. This approach could be expanded to allow for some flexibility in this respect. At the moment, these lyrics need to be provided manually. In the future, those could be retrieved automatically (for example by using the approaches described above).

8.5 Conclusion

In this section, three approaches to lyrics-to-audio alignment and retrieval and one practical application were presented. The first one uses HMMs trained on speech for alignment. The mean alignment error in the 2017 *MIREX* challenge was 5.11s on unaccompanied singing, and 13.96s and 17.7s on two polyphonic data sets. A manual check on some examples suggests that the algorithm is already usable in practice in many cases, even for polyphonic music. It was also used to generate the annotations for the *DAMP* training data sets as described in section 5.3.

The second algorithm is based on Dynamic Time Warping. In its first step, phoneme posteriorgrams are extracted with the various acoustic models described in chapter 5, in particular those trained on *TIMIT* speech data, the time-stretched and pitch-shifted version *TimitM*, and several of the *DAMP*-based data sets whose annotations were generated with the previous approach. Then, a one-hot binary template is generated from the lyrics to be aligned, and an optimal alignment between this and the posteriorgram is computed via DTW. In the *MIREX* challenge, this approach achieved a mean alignment error of 9.77s on the unaccompanied singing data, and of 27.94s and 22.23s on the two polyphonic data sets.
The same approach can also be used for retrieving lyrics from a text database with a sung query. To this end, binary templates are generated for all possible lyrics, and the alignment is performed for all of them. Then, the result with the lowest DTW cost is selected as the winner. When the whole song is used as the input, an accuracy of 86% for the single best result is obtained. If the 10 best results are taken into account, this rises to 91%. When using only short sung lines as input, the mean Top-1 accuracy for retrieving the correct song lyrics of the whole song is 37%. For the best 100 results, the accuracy is 67%. An interesting result was the difference between the female and the male test sets: On the female test set, retrieval with models trained on speech was significantly lower than on the male set (39% vs. 58% on the song-wise

task). This may happen because the frequency range of female singing is not covered well with speech data only. When using acoustic models trained on singing, this difference disappears and the results become significantly higher in general. Even for a model trained on less data than that contained in *TIMIT*, the average accuracy is 82%.

The third approach is also based on posteriorgrams generated with the models from chapter 5. However, instead of operating directly on them, phoneme sequences are extracted. This is done by selecting the phoneme with the highest probability in each frame and compressing this sequence down to discard unlikely phoneme occurrences. This process takes the known phoneme confusions of the classifier into account. Several improvement steps were added. The extracted phoneme sequence is then aligned to the expected one (from the known lyrics) using the Levenshtein distance. In the *MIREX* challenge, this algorithm produced the best results with a mean error of 2.87s on unaccompanied singing, and 7.34s and 4.55s on polyphonic music.
For lyrics retrieval, the extracted phoneme sequence is compared to all the sequences in the lyrics database using the Levenshtein distance, and the result with the lowest distance is selected. On the same test data as above, the approach achieves an accuracy of 94% for whole-song inputs. With line-wise inputs, the accuracy is 54% for the Top-1 result, and 88% for the Top-100. Once again, there is a significant difference between the female and male results with models trained on speech, which disappears with singing-based models.
Tests on a smaller database of hand-selected clean audio samples resulted in much higher accuracies for single-line queries. This suggests that the system can perform much better when provided with clean inputs; this would be feasible in a real-world application.

Finally, the HMM and DTW alignments were tested in an application scenario: The extraction of expletives from popular music. To this end, known lyrics were aligned to polyphonic song recordings containing such words. Then, the found time segments were masked with other sounds, or stereo subtraction was performed to remove the singing voice. At a tolerance of 1s, 92% of the expletives were detected in their correct positions, making the system usable for practical applications.

In all experiments, analysis of the errors showed the same possible sources as described in section 5.3.4. Many of them had to do with enunciation issues (clarity, accents, or children's voices) or issues with the recording itself (background music,

clipping, extraneous speaking). These problems would not be as prevalent in professional recordings. However, some of them could be fixed with adaptations to the algorithm. In polyphonic alignment experiments, errors also occurred due to prominent instruments, in particular during instrumental solos. As described in section 3.1, this is also an issue in other MIR tasks related to singing, such as Vocal Activity Detection.

As described, the developed algorithms are in many cases ready for practical applications. In the future, it would be interesting to see them integrated into existing systems. There are also ways to make them more robust, e.g. to errors in the lyrics transcriptions or to mistakes or variations by the singers. No special steps have been taken so far to adapt them to polyphonic music; Vocal Activity Detection or source separation could be performed in the pre-processing.
So far, the retrieval approaches were only tested on a relatively small lyrics database containing 12,000 lines of lyrics across 300 songs. For larger databases, scalability can become an issue. One partial solution was already integrated into the phoneme-based method. To take this one step further, both retrieval algorithms could first perform a rough search with smaller lyrical "hashes" to find possible matches, and then perform a refinement as presented. This is similar to techniques that are already used in audio fingerprinting [181].
Alternatively, the phoneme-based approach could be expanded to retrieve lyrics from the internet instead of from a fixed database, e.g. with Semantic Web techniques.

9 Conclusion

9.1 Summary

In this work, ASR algorithms for various tasks were applied and adapted to singing. Five such tasks were considered: Phoneme recognition, language identification, keyword spotting, lyrics-to-audio alignment, and lyrics retrieval.

Two strategies for improving speech recognition technologies for singing in general were identified: Training better-matching acoustic models for phoneme recognition, and making specific algorithms more robust to singing - either by balancing possible deficits that occur because of the singing-specific characteristics, or by exploiting knowledge about sung vocal production.

The main bottleneck in almost all tasks is the **recognition of phonemes** in singing. Three approaches for this were tested: Recognition with models trained on speech, recognition with models trained on speech that was made more "song-like" ("songified"), and recognition with models trained on singing. The main issue in this area of research is the lack of large training data sets of singing with phoneme annotations. For this reason, the "songified" approach was developed, which slightly but significantly improved results over the models trained on pure speech. For training on actual singing data, the large *DAMP* data set of unaccompanied amateur singing recordings was chosen. Phonemes obtained from the matching textual lyrics were automatically aligned to these recordings, and new acoustic models were trained on the resulting annotated data set. These models showed large improvements for phoneme recognition in singing.

This training strategy is the main contribution of this work, and forms the basis for all the subsequent tasks. As proposed in the introduction, various algorithms for the individual tasks were then tested and adapted by improving robustness when applied to singing instead of speech, or by taking useful knowledge about singing into account. Robustness was, for example, increased by using i-vector extraction for language iden-

tification, by employing keyword-filler HMMs to keyword spotting, and by adapting alignment approaches to enable varying phoneme durations. Singing characteristics are exploited, for example, by basing language identification methods on long recordings, by integrating phoneme duration knowledge into keyword spotting, and by including known phoneme confusions in singing into the alignment and retrieval algorithms.
The following paragraphs will sum up the individual adaptations to the tasks in detail:

For **language identification** in singing, the state-of-the-art i-vector approach from ASR was tested and produced good results. It is especially suited to singing because it implements a strategy for removing irrelevant and repetitive information, such as signal components caused by the channel. These results were shown to improve to be practically usable when long recordings are available, which is usually the case for singing. In addition to this method, a new approach based on phoneme statistics was developed. In this approach, phoneme posteriorgrams are first generated with the models trained in the phoneme recognition experiments, then statistics are calculated, and different models are trained on those. This approach does not perform quite as well as the i-vector method, but is easier to implement under certain circumstances (when phoneme recognition needs to be performed anyway). Long recordings are necessary to calculate meaningful statistics, but, once again, this is not a problem on singing data.

Keyword spotting is a task that has not frequently been a subject of research, and there is still a lot of room for improvement. In this work, keyword-filler HMMs were tested. In contrast to other state-of-the-art methods, this approach does not rely on highly stable phoneme durations, which are very common in speech, but not in singing. The method detects keywords with high recall, but often poor precision. Knowledge about probable phoneme durations was obtained from analyzing sung recordings and added in a post-processing step. This improved results for frequently-occurring keywords. The performance may not be good enough for practical applications yet, but these approaches show promise for future research. The length of the keyword (number of phonemes) is a major deciding factor. Searching for long keywords or phrases is already possible in practice, and this is a probable usage scenario for music collections.

Lyrics-to-audio alignment is a relatively well-researched topic. In this work, the new acoustic models were also applied to this task, alongside a traditional HMM-based approach. Two methods for utilizing phoneme posteriorgrams for alignment were developed. The first one calculates a DTW between the posteriorgram and a binary

representation of the expected lyrics. The resulting path is the alignment. Since no knowledge about the duration of the constituting phonemes is available in the text, and often varies widely in singing, the algorithm is modified to not punish such varying durations. The second approach first extracts a plausible phoneme sequence from the posteriorgram, taking known phoneme confusions in singing into account and compressing extended productions of similar phonemes. Then, the Levenshtein distance between this symbolic (phonetic) representation and the phonemes obtained from the textual lyrics is calculated, and the alignment path is the result.
The HMM and DTW approaches perform acceptably, while the Levenshtein approach produces much better results. These algorithms are suited for practical applications. One such example was described: The automatic detection and removal of expletives in songs.

Lyrics retrieval from a textual database based on sung queries is performed with the same algorithms as the alignment. Alignment is performed between the posteriorgram generated from the audio and all possible lyrics. Then, the ones with the best scores/lowest distances are selected as the result.
Once again, the DTW approach performs feasibly, while the Levenshtein approach produces very good results that are practically usable, even on short queries. Additionally, it allows for fast search and optimizations on large lyrics databases, which is a so-far underresearched use case.

9.2 Contributions

Major research contributions of this work include:

A new large data set of phonetic annotations for unaccompanied singing As described in sections 4.2.3 and 5.3, the pre-existing *DAMP* data set of unaccompanied singing was used as a basis for this. The matching textual lyrics were scraped from the internet, and automatically aligned to the audio with an HMM model trained on speech. Various strategies were tested for this. The resulting phoneme annotations are not 100% accurate, but the data set is very large and these inaccuracies balance out, as the experiments building on the data demonstrate. Various configurations for this data set were composed for different purposes, including female and male singing test sets.

Acoustic models trained on this data set New acoustic models (DNNs) were then trained on the generated training data sets (section 5.3). The performance of these models was thoroughly compared to others, and they generally performed better when applied to singing phoneme recognition. They were also integrated into the other tasks (language identification, keyword spotting, lyrics-to-audio alignment and retrieval) and demonstrated improvements in almost all cases.

A novel approach for language identification based on phoneme statistics In addition to an i-vector-based approach from ASR, a completely new method for language identification was developed on the basis of phoneme posteriorgrams. This approach is particularly suited to singing because long recordings are often available in music applications. As described in section 6.2, phoneme statistics are calculated on these posteriorgrams over long time frames (e.g. whole songs), and then new models are trained on these statistics. Unseen recordings can then be subjected to the same process to determine their languages. The results of this approach were not quite as good as those of the i-vector algorithm, but they show promise. This method is easier to implement than the one using i-vectors when long audio sequences are available, and when phoneme posteriorgrams need to be extracted from them for other purposes.

A new method for flexibly integrating knowledge about phoneme durations into keyword spotting The presented approach for keyword spotting is based on keyword-filler HMMs. In order to improve its results, the algorithm was expanded to take a-priori knowledge about the plausible durations of individual phonemes into account (see section 7.2). This was implemented via post-processor duration modeling: The model is tuned to return many results (resulting in a high recall), and then those with improbable phoneme durations are discarded to improve the precision. Influence of individual phoneme durations is easily turned on or off. This form of duration modeling has not been applied to keyword-filler HMMs before.

A novel method for extracting plausible phoneme sequences from posteriorgrams The described phoneme recognition approaches result in phoneme posteriorgrams, and many algorithms operate directly on them. However, there are also applications where a fixed phoneme sequence is required. In section 8.3, a new method for extracting such sequences from posteriorgrams is presented. Traditionally, HMMs would be used for this task, but they have a number of disadvantages in the use case at hand. The

results of this phoneme extraction method are further used in this work for alignment and retrieval.

Two new approaches for lyrics-to-audio alignment In addition to classic HMM-based lyrics-to-audio alignment, this thesis presents two novel approaches that operate on posteriorgrams: One based on DTW, and one based on Levenshtein distance calculation. This enables them to make use of the new acoustic models. The developed algorithms were submitted to the *MIREX* 2017 challenge for lyrics-to-audio alignment, where the Levenshtein-based approach outperformed the other submissions.

Two first approaches to lyrics retrieval based purely on sung queries The same two approaches for lyrics-to-audio alignment can be applied to the task of lyrics retrieval. This is an application that has only rarely been the subject of research so far. This thesis describes the implementation of the presented methods for retrieval of textual song lyrics from a database with sung queries.

9.3 Limitations and future work

Suggestions for the individual tasks and approaches are described in the corresponding chapters. This section focuses on the over-all field of ASR for singing.
In general, phoneme recognition is still the major bottleneck for many of the described tasks. This thesis shows possible ways to improve this recognition. However, results could still be much higher. Future research approaches may employ even larger data sets of singing for training, preferably with more reliable phonetic annotations (e.g. manual annotations). Alternatively, models could be trained in an unsupervised or semi-supervised way with large amounts of unannotated data and small sets of reliably annotated data. New machine learning techniques could be tested as well, such as CNNs, RNNs, or end-to-end models, or methods for unsupervised pre-training like autoencoders.
Unusual or unclear pronunciations, accents, and unusual voices (e.g. children's voices) also lead to unsatisfactory results in many tasks. New models trained on a larger variety of data would improve robustness. The same applies to problems with the audio quality or channels, and recording conditions not seen during training.

Some of the algorithms require exact annotations (e.g. alignment and retrieval). This is not always a given in a real-world scenario, where singers will perform different

words, additional vocalizations, unexpected repetitions etc. The actual calculations could be made robust to such changes.

Most of the presented approaches have so far only been applied to unaccompanied singing. A major step forward would be the adaptation to polyphonic music. This was already done for alignment and retrieval, but could be improved significantly. There are several possible routes that future research could take. First, acoustic models could be trained on accompanied singing instead. This would require large amounts of realistic training data, and probably more sophisticated models in order to represent the more complex structures. Second, source separation could be integrated to extract the singing track from the audio, and only perform the analysis on this part. As a third alternative, Vocal Activity Detection could be applied beforehand to only analyze the segments where actual singing occurs. In the presented experiments, instrumental solos in particular frequently led to misalignments or recognition errors.

Finally, the developed algorithms could be employed for the applications described in section 1.1. It would be highly interesting to see them integrated into other MIR systems, e.g. for genre or regional classification or for mood detection. The other practical scenarios could also make good use of them, for example in karaoke systems, in audio identification, or in cover song detection. Search algorithms or similarity calculation based on the analyzed characteristics could be used in practical MIR systems.

Bibliography

[1] J. S. Downie, "Music Information Retrieval," in *Annual Review of Information Science and Technology*, B. Cronin, Ed., chapter 7, pp. 295–340. Information Today, 2003.

[2] B. H. Juang and L. R. Rabiner, *Encyclopedia of Language and Linguistics*, chapter Automatic Speech Recognition – A Brief History of the Technology Development, Elsevier, 2nd edition, 2005.

[3] A. Loscos, P. Cano, and J. Bonada, "Low-delay singing voice alignment to text," in *International Computer Music Conference (ICMC)*, 1999.

[4] A. Kruspe, H. Lukashevich, J. Abesser, H. Grossmann, and C. Dittmar, "Automatic classification of musical pieces into global cultural areas," in *AES 42nd International Conference on Semantic Audio*, 2011.

[5] J. Aucouturier and F. Pachet, "Improving timbre similarity: How high is the sky?," in *Journal of Negative Results in Speech and Audio Sciences*, 2004, vol. 1.

[6] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," in *arXiv*, 2016, `https://arxiv.org/abs/1609.03499`.

[7] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543–565, May 2012.

[8] M. Müller, *Information Retrieval for Music and Motion*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[9] P. Mermelstein, "Distance measures for speech recognition: Psychological and instrumental," in *Pattern Recognition and Artificial Intelligence*, C. H. Chen, Ed., pp. 374–388. Academic Press, 1976.

[10] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-28, no. 4, pp. 357–366, 1980.

[11] J. S. Bridle and M. D. Brown, "An experimental automatic word recognition system," Tech. Rep., Joint Speech Research Unit, Ruislip, England, 1974.

[12] D. D. O'Shaughnessy, "Invited paper: Automatic speech recognition: History, methods and challenges," *Pattern Recognition*, vol. 41, no. 10, pp. 2965–2979, 2008.

[13] B. Bielefeld, "Language identification using Shifted Delta Cepstrum," in *Annual Speech Research Symposium*, 1994.

[14] P. A. Torres-Carrasquillo, E. Singer, M. A. Kohler, R. J. Greene, D. A. Reynolds, and J. J. R. Deller, "Approaches to language identification using Gaussian mixture models and Shifted Delta Cepstral features," in *International Conference on Spoken Language Processing (ICSLP)*, 2002.

[15] W. M. Campbell, J. P. Campbell, D. A. Reynolds, E. Singer, and P. A. Torres-Carrasquillo, "Support vector machines for speaker and language recognition," *Computer Speech and Language*, vol. 20, pp. 210–229, 2006.

[16] F. Allen, E. Ambikairajah, and J. Epps, "Language identification using warping and the shifted delta cepstrum," in *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, 2005.

[17] H. Hermansky, "Perceptual linear predictive (PLP) analysis of speech," *Journal of the Acoustical Society of America (JASA)*, vol. 57, no. 4, pp. 1738–52, Apr. 1990.

[18] F. Hönig, G. Stemmer, C. Hacker, and F. Brugnara, "Revising perceptual linear prediction (PLP)," in *Interspeech*, 2005.

[19] P. C. Woodland, M. J. F. Gales, and D. Pye, "Improving environmental robustness in large vocabulary speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1996.

[20] J. Makhoul and L. Cosell, "LPCW: An LPC vocoder with linear predictive spectral warping," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1976.

[21] S. S. Stevens, "On the psychophysical law," *Psychological Review*, vol. 64, no. 3, pp. 153–181, 1957.

[22] J. Makhoul, "Spectral linear prediction: Properties and applications," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 23, no. 3, pp. 283–296, 1975.

[23] H. Hermansky, N. Morgan, A. Bayya, and P. Kohn, "Rasta-plp speech analysis," Tech. Rep. TR-91-069, ICSI, 1991.

[24] H. Hermansky and S. Sharma, "TRAPs – classifiers of temporal patterns," in *International Conference on Spoken Language Processing (ICSLP)*, 1998.

[25] H. Hermansky and S. Sharma, "Temporal patterns (TRAPs) in ASR of noisy speech," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1999.

[26] P. Matejka, I. Szöke, P. Schwarz, and J. Cernocky, "Automatic language identification using phoneme and automatically derived unit strings," in *International Conference on Text, Speech and Dialogue (TSD)*, 2004.

[27] J. K. Hansen, "Recognition of phonemes in a-cappella recordings using temporal patterns and mel frequency cepstral coefficients," in *Sound and Music Computing Conference (SMC)*, 2012.

[28] J. Li, L. Deng, Y. Gong, and R. Hï¿½b-Umbach, *Robust Automatic Speech Recognition: A Bridge to Practical Applications*, Academic Press, 2015.

[29] L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[30] D. P. W. Ellis, "Dynamic Time Warp (DTW) in Matlab," 2003, `http://www.ee.columbia.edu/~dpwe/resources/matlab/pvoc/`, Web resource.

[31] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics – Doklady*, vol. 10, no. 8, 1966.

[32] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, Mar. 2001.

[33] R. Lee, Ed., *Applied Computing and Information Technology*, Studies in Computational Intelligence. Springer, 2017.

[34] D. Jurafsky, "Minimum Edit Distance," University Lecture, 2014.

[35] M. J. Hunt, "Figures of merit for assessing connected-word recognisers.," *Speech Communication*, vol. 9, no. 4, pp. 329–336, 1990.

[36] J. Schmidhuber, "Deep learning in Neural Networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

[37] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[38] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 1, pp. 72–83, Jan. 1995.

[39] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York, NY, USA, 3rd edition, 2007.

[40] H. Permuter, J. Francos, and I. Jermyn, "Gaussian mixture models of texture and colour for image database retrieval," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2003.

[41] C. Alexander, "Normal mixture diffusion with uncertain volatility: Modelling short- and long-term smile effects," *Journal of Banking and Finance*, vol. 28, no. 12, pp. 2957–2980, 2004.

[42] J. Chen, O. E. Adebomi, O. S. Olusayo, and W. Kulesza, "The evaluation of the Gaussian Mixture Probability Hypothesis Density approach for multi-target tracking," in *IEEE International Conference on Imaging Systems and Techniques (IST)*, 2010.

[43] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[44] L. E. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bulletin of the American Mathematical Society*, vol. 73, no. 3, pp. 360–363, May 1967.

[45] L. E. Baum and G. R. Sell, "Growth transformations for functions on manifolds.," *Pacific Journal of Mathematics*, vol. 27, no. 2, pp. 211–227, 1968.

[46] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[47] L. E. Baum, "An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process," *Inequalities*, vol. 3, pp. 1–8, 1972.

[48] L. R. Rabiner and B. H. Juang, "An introduction to Hidden Markov Models," *IEEE Acoustics, Speech, and Signal Processing (ASSP) Magazine*, Jan. 1986.

[49] L. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition," *Proceedings of the IEEE*, 1989.

[50] F. Jelinek, L. R. Bahl, and R. L. Mercer, "Design of a linguistic statistical decoder for the recognition of continuous speech," *IEEE Transactions on Information Theory*, vol. 21, no. 3, pp. 250–256, 1975.

[51] M. Gales and S. Young, "The application of Hidden Markov Models in speech recognition," *Foundations and Trends in Signal Processing*, vol. 1, no. 3, pp. 195–304, Jan. 2007.

[52] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, MA, USA, 1999.

[53] R. Nag, K. Wong, and F. Fallside, "Script recognition using Hidden Markov Models," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1986.

[54] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler, "Hidden Markov Models in computational biology: Applications to protein modeling," *Journal of Molecular Biology*, vol. 235, pp. 1501–1531, Feb. 1994.

[55] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis*, Cambridge University Press, 1998.

[56] C. M. Bishop, *Pattern Recognition and Machine Learning*, Information Science and Statistics. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[57] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to Support Vector Classification," Tech. Rep., National Taiwan University, 2010.

[58] D. Boswell, "Introduction to Support Vector Machines," 2002.

[59] C. J. C. Burges, "A tutorial on Support Vector Machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[60] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-End Factor Analysis for Speaker Verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, May 2011.

[61] D. Martinez, O. Plchot, and L. Burget, "Language Recognition in iVectors Space," in *Interspeech*, 2011.

[62] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted Gaussian Mixture Models," *Digital Signal Processing*, vol. 10, no. 1–3, pp. 19–41, Jan. 2000.

[63] C. Charbuillet, D. Tardieu, and G. Peeters, "GMM Supervector for content based music similarity," in *Conference on Digital Audio Effects (DAFx)*, 2011.

[64] W. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[65] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 5, pp. 386–408, 1958.

[66] A. G. Ivakhnenko and V. G. Lapa, "Cybernetic Predicting Devices," Tech. Rep., CCM Information Corporation, 1965.

[67] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.

[68] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. thesis, Harvard University, 1974.

[69] A. H. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, 1989.

[70] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. S. Kruthiventi, and R. V. Babu, "A taxonomy of Deep Convolutional Neural Nets for computer vision," *Frontiers in Robotics and AI*, vol. 2, no. 36, 2016.

[71] Y. Goldberg, "A primer on Neural Network models for natural language processing," *Journal of Artificial Intelligence Research (JAIR)*, vol. 57, pp. 345–420, 2016.

[72] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[73] J. Markoff, "Scientists see promise in Deep Learning programs," *The New York Times*, Nov. 23, 2012.

[74] L. Deng, G. Hinton, and B. Kingsbury, "New types of Deep Neural Network learning for speech recognition and related applications: An overview," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[75] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms*, Cambridge University Press, New York, NY, USA, 2002.

[76] R. Campbell, "Demystifying Deep Neural Nets," *Medium*, Mar. 31, 2017, `https://medium.com/manchester-futurists/demystifying-deep-neural-nets-efb726eae941`.

[77] S. Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[78] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–80, 1997.

[79] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[80] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory Recurrent Neural Network architectures for large scale acoustic modeling," in *Interspeech*, 2014.

[81] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2012.

[82] K. J. Piczak, "Environmental sound classification with Convolutional Neural Networks," in *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2015.

[83] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional Neural Networks for speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[84] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[85] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of Deep Networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2006.

[86] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *International Conference on Machine Learning (ICML)*, 2009.

[87] D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column Deep Neural Network for traffic sign classification," *Neural Networks*, vol. 32, pp. 333–338, 2012.

[88] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training Deep architectures and the effect of unsupervised pre-training," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[89] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[90] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. D. Shet, "Multi-digit number recognition from street view imagery using Deep Convolutional Neural Networks," in *International Conference on Learning Representations*, 2014.

[91] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with Deep Convolutional Generative Adversarial Networks," in *arXiv*, 2015, `https://arxiv.org/abs/1511.06434`.

[92] H. Fujihara and M. Goto, "Lyrics-to-audio alignment and its applications," in *Multimodal Music Processing*, M. Müller, M. Goto, and M. Schedl, Eds., vol. 3. Dagstuhl Follow-Ups, 2012.

[93] A. M. Kruspe, "Keyword spotting in a-capella singing," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2014.

[94] J. Schlüter, "Learning to Pinpoint Singing Voice from Weakly Labeled Examples," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2016.

[95] C.-K. Wang, R.-Y. Lyu, and Y.-C. Chiang, "An automatic singing transcription system with multilingual singing lyric recognizer and robust melody tracker," in *Interspeech*, 2003.

[96] T. Hosoya, M. Suzuki, A. Ito, and S. Makino, "Lyrics recognition from a singing voice based on finite state automaton for music information retrieval," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2005.

[97] M. J. F. Gales and P. C. Woodland, "Mean and variance adaptation within the MLLR framework," *Computer Speech & Language*, vol. 10, no. 4, pp. 249–264, 1996.

[98] M. Gruhne, K. Schmidt, and C. Dittmar, "Phoneme recognition in popular music," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2007.

[99] M. Gruhne, K. Schmidt, and C. Dittmar, "Detecting phonemes within the singing of polyphonic music," in *International Conference on Music Communication Science (ICoMCS)*, 2007.

[100] K. Dressler, "Sinusoidal Extraction using an efficient implementation of a multi-resolution FFT," in *Conference on Digital Audio Effects (DAFx)*, 2006.

[101] A. Härmä and U. K. Laine, "A comparison of warped and conventional Linear Predictive Coding," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 5, pp. 579–588, 2001.

[102] H. Fujihara, T. Kitahara, M. Goto, K. Komatani, T. Ogata, and H. Okuno, "Singer identification based on accompaniment sound reduction and reliable frame selection," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2005.

[103] G. Szepannek, M. Gruhne, B. Bischl, S. Krey, T. Harczos, F. Klefenz, C. Dittmar, and C. Weihs, *Classification as a tool for research*, chapter "Perceptually Based Phoneme Recognition in Popular Music", Springer, Heidelberg, 2010.

[104] H. Fujihara, M. Goto, and H. G. Okuno, "A novel framework for recognizing phonemes of singing voice in polyphonic music," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2009.

[105] G. Fant, "The source filter concept in voice production," *KTH Dept. for Speech, Music and Hearing: Quarterly Progress and Status Report (STL-QPSR)*, vol. 22, no. 1, pp. 21–37, 1981.

[106] A. Mesaros and T. Virtanen, "Adaptation of a speech recognizer for singing voice," in *European Signal Processing Conference (EUSIPCO)*, 2009.

[107] A. Mesaros and T. Virtanen, "Recognition of phonemes and words in singing," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2010.

[108] A. Mesaros and T. Virtanen, "Automatic understanding of lyrics from singing," *Akustikkapäivät*, pp. 1–6, 2011.

[109] T. Virtanen, A. Mesaros, and M. Ryynänen, "Combining pitch-based inference and non-negative spectrogram factorization in separating vocals from polyphonic music," in *ISCA Tutorial and Research Workshop on Statistical and Perceptual Audition (SAPA)*, 2008.

[110] A. Mesaros and T. Virtanen, "Automatic Recognition of Lyrics in Singing," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2010, pp. 1–11, 2010.

[111] M. McVicar, D. P. W. Ellis, and M. Goto, "Leveraging repetition for improved automatic lyric transcription in popular music," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[112] J. G. Fiscus, "A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER)," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 1997.

[113] M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka, "RWC Music Database: Popular, Classical, and Jazz Music Databases," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2002.

[114] A. Weber and R. Smits, "Consonant and vowel confusion patterns by American English listeners," in *International Congress of Phonetic Sciences*, 2003.

[115] B. T. Meyer, M. Wächter, T. Brand, and B. Kollmeier, "Phoneme confusions in human and automatic speech recognition," in *Interspeech*, 2007.

[116] H. Hollien, A. R. Mendes-Schwartz, and K. Nielsen, "Perceptual confusions of high-pitched sung vowels," *Journal of Voice*, vol. 14, no. 2, pp. 287–298, 2000.

[117] Y. Wang, M.-Y. Kan, T. L. Nwe, A. Shenoy, and J. Yin, "LyricAlly: Automatic synchronization of acoustic musical signals and textual lyrics," in *ACM International Conference on Multimedia*, 2004.

[118] M. Kan, Y. Wang, D. Iskandar, T. L. Nwe, and A. Shenoy, "LyricAlly: Automatic Synchronization of Textual Lyrics to Acoustic Music Signals," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 2, pp. 338–349, 2008.

[119] D. Iskandar, Y. Wang, M.-Y. Kan, and H. Li, "Syllabic level automatic synchronization of music signals and text lyrics," in *ACM International Conference on Multimedia*, 2006.

[120] A. Sasou, M. Goto, S. Hayamizu, and K. Tanaka, "An auto-regressive, non-stationary excited signal parameter estimation method and an evaluation of a singing-voice recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2005.

[121] K. Chen, S. Gao, Y. Zhu, and Q. Sun, "Popular song and lyrics synchronization and its application to music information retrieval," in *SPIE Multimedia Computing and Networking*, 2006.

[122] H. Fujihara, M. Goto, J. Ogata, K. Komatani, T. Ogata, and H. G. Okuno, "Automatic synchronization between lyrics and music CD recordings based on Viterbi alignment of segregated vocal signals," in *IEEE International Symposium on Multimedia (ISM)*, 2006.

[123] H. Fujihara and M. Goto, "Three techniques for improving automatic synchronization between music and lyrics: Fricative detection, filler model, and novel feature vectors for vocal activity detection," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2008.

[124] H. Fujihara, M. Goto, J. Ogata, and H. G. Okuno, "LyricSynchronizer: Automatic Synchronization System Between Musical Audio Signals and Lyrics," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 6, pp. 1252–1261, 2011.

[125] M. Mauch, H. Fujihara, and M. Goto, "Lyrics-to-audio alignment and phrase-level segmentation using incomplete internet-style chord annotations," in *Sound and Music Computing Conference (SMC)*, 2010.

[126] M. Mauch, H. Fujihara, and M. Goto, "Integrating additional chord information into hmm-based lyrics-to-audio alignment," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 200–210, 2012.

[127] C. H. Wong, W. M. Szeto, and K. H. Wong, "Automatic lyrics alignment for Cantonese popular music," *Multimedia Systems*, vol. 12, no. 4–5, pp. 307–323, 2007.

[128] K. Lee and M. Cremer, "Segmentation-based lyrics-audio alignment using dynamic programming," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2008.

[129] A. Mesaros and T. Virtanen, "Automatic alignment of music audio and lyrics," in *Conference on Digital Audio Effects (DAFx)*, 2008.

[130] R. Gong, P. Cuvillier, N. Obin, and A. Cont, "Real-time audio-to-score alignment of singing voice based on melody and lyric information," in *Interspeech*, 2015.

[131] G. Dzhambazov, A. Srinivasamurthy, S. Sentürk, and X. Serra, "On the use of note onsets for improved lyrics-to-audio alignment in Turkish Makam music," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2016.

[132] M. Suzuki, T. Hosoya, A. Ito, and S. Makino, "Music information retrieval from a singing voice based on verification of recognized hypotheses," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2006.

[133] M. Suzuki, T. Hosoya, A. Ito, and S. Makino, "Music information retrieval from a singing voice using lyrics and melody information," *EURASIP Journal on Advances in Signal Processing*, 2007.

[134] C.-C. Wang, J.-S. R. Jang, and W. Wang, "An Improved Query by Singing/Humming System Using Melody and Lyrics Information," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2010.

[135] A. Mesaros and T. Virtanen, "Recognition of phonemes and words in singing," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2010.

[136] W.-H. Tsai and H.-M. Wang, "Towards Automatic Identification Of Singing Language In Popular Music Recordings," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2004.

[137] J. Schwenninger, R. Brueckner, D. Willett, and M. E. Hennecke, "Language Identification in Vocal Music," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2006.

[138] V. Chandraskehar, M. E. Sargin, and D. A. Ross, "Automatic language identification in music videos with low level audio and visual features," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[139] H. Fujihara, M. Goto, and J. Ogata, "Hyperlinking lyrics: A method for creating hyperlinks between phrases in song lyrics," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2008.

[140] G. Dzhambazov, S. Sentürk, and X. Serra, "Searching lyrical phrases in a-capella Turkish Makam recordings," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2015.

[141] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue, "TIMIT Acoustic-Phonetic Continuous Speech Corpus," Tech. Rep., Linguistic Data Consortium, Philadelphia, 1993.

[142] V. Zue, S. Seneff, and J. Glass, "Speech database development at MIT: Timit and beyond," *Speech Communication*, vol. 9, no. 4, pp. 351–356, 1990.

[143] K.-F. Lee and H.-W. Hon, "Speaker-independent phone recognition using Hidden Markov Models," *IEEE Transactions on Acoustics, Speech, and Signal processing*, vol. 37, no. 11, pp. 1641–1648, 1989.

[144] A. Martin and M. Pryzbocki, "2003 NIST Language Recognition Evaluation," Tech. Rep., Linguistic Data Consortium, Philadelphia, 2006.

[145] J. C. Smith, *Correlation analyses of encoded music performance*, Ph.D. thesis, Stanford University, 2013.

[146] S. J. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, *The HTK Book Version 3.4*, Cambridge University Press, 2006.

[147] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," in *arXiv*, 2016, `https://arxiv.org/abs/1605.02688`.

[148] C. Jankowski, A. Kalyanswamy, S. Basson, and J. Spitz, "NTIMIT: A phonetically balanced, continuous speech telephone bandwidth speech database," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1990.

[149] H.-G. Hirsch and D. Pearce, "The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions," in *Interspeech*, 2000.

[150] L. Pérez and J. M. Wang, "The effectiveness of data augmentation in image classification using deep learning," in *arXiv*, 2017, `https://arxiv.org/abs/1712.04621`.

[151] B. McFee, E. Humphrey, and J. Bello, "A software framework for musical data augmentation," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2015.

[152] D. P. W. Ellis, "A phase vocoder in Matlab," 2002, `http://www.ee.columbia.edu/~dpwe/resources/matlab/dtw/`, Web resource.

[153] J. L. Flanagan and R. M. Golden, "Phase vocoder," *Bell System Technical Journal*, pp. 1493–1509, Nov. 1966.

[154] M. Dolson, "The phase vocoder: A tutorial," *Computer Music Journal*, vol. 10, no. 4, pp. 14–27, 1986.

[155] C. Arft, "AutoTune Toy," 2010, `http://www.mathworks.com/matlabcentral/fileexchange/26337-autotune-toy`.

[156] D. Jurafsky and J. H. Martin, *Speech and language processing: An introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice Hall, 2009.

[157] M. A. Zissman, "Comparison of four approaches to automatic language identification of telephone speech," *IEEE Transactions on Speech and Audio Processing*, vol. 4, no. 1, pp. 31–44, Jan. 1996.

[158] H. Li and B. Ma, "A phonotactic language model for spoken language identification," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 2005, ACL '05.

[159] P. Matejka, P. Schwarz, J. Cernocky, and P. Chytil, "Phonotactic language identification using high quality phoneme recognition," in *Interspeech*, 2005.

[160] E. Singer, P. A. Torres-Carrasquillo, T. P. Gleason, W. M. Campbell, and D. A. Reynolds, "Acoustic, phonetic, and discriminative approaches to automatic language identification," in *Eurospeech*, 2003.

[161] K. M. Berkling, *Automatic Language Identification with Sequences of Language-Independent Phoneme Clusters*, Ph.D. thesis, Oregon Graduate Institute of Science & Technology, 1996.

[162] H. Li, B. Ma, and C.-H. Lee, "A Vector Space Modeling Approach to Spoken Language Identification," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 15, pp. 271–284, 2007.

[163] M. Peche, M. H. Davel, and E. Barnard, "Phonotactic spoken language identification with limited training data," in *Interspeech*, 2007.

[164] "The 2015 NIST Language Recognition Evaluation Plan (LRE15)," Tech. Rep., NIST, 2015.

[165] H. Eghbal-zadeh, B. Lehner, M. Schedl, and G. Widmer, "i-vectors for timbre-based music similarity and music artist classification," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2015.

[166] H. Eghbal-zadeh, M. Schedl, and G. Widmer, "Timbral modeling for music artist recognition using i-vectors," in *European Signal Processing Conference (EUSIPCO)*, 2015.

[167] A. Martin and C. Greenberg, "The 2009 NIST Language Recognition Evaluation," in *Odyssey. The Speaker and Language Recognition Workshop*, 2010.

[168] A. M. Kruspe, "Improving singing language identification through i-vector extraction," in *Proceedings of the 17th International Conference on Digital Audio Effects (DAFx-14)*, 2014.

[169] A. Moyal, V. Aharonson, E. Tetariy, and M. Gishri, *Phonetic Search Methods for Large Speech Databases*, chapter 2: Keyword spotting methods, Springer, 2013.

[170] I. Szöke, P. Schwarz, P. Matejka, L. Burget, M. Karafiat, and J. Cernocky, "Phoneme based acoustics keyword spotting in informal continuous speech.," in *International Conference on Text, Speech and Dialogue (TSD)*, 2005.

[171] A. Jansen and P. Niyogi, "An experimental evaluation of keyword-filler Hidden Markov Models," Tech. Rep., 2009.

[172] E. I. Chang and R. P. Lippmann, "Figure of merit training for detection and spotting," in *International Conference on Neural Information Processing Systems (NIPS)*, 1993.

[173] A. J. K. Thambiratnam, *Acoustic keyword spotting in speech with applications to data mining*, Ph.D. thesis, Queensland University of Technology, 2005.

[174] J. D. Ferguson, "Variable duration models for speech," in *Symposium on the Application of Hidden Markov Models to Text and Speech*, 1980.

[175] B. H. Juang, L. R. Rabiner, S. E. Levinson, and M. M. Sondhi, "Recent developments in the application of Hidden Markov Models to speaker-independent isolated word recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1985.

[176] S. E. Levinson, "Continuously variable duration Hidden Markov Models for speech analysis," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1986.

[177] D. Burshtein, "Robust parametric modeling of durations in Hidden Markov Models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 4, no. 3, pp. 240–242, 1996.

[178] A. M. Kruspe, "Keyword spotting in a-capella singing with duration-modeled HMMs," in *European Signal Processing Conference (EUSIPCO)*, 2015.

[179] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[180] J. Sundberg, "Perception of singing," in *The Psychology of Music*, D. Deutsch, Ed. Academic Press, 3rd edition, 2012.

[181] A. L. Wang, "An industrial-strength audio search algorithm," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2003.

List of Figures