Technische Universität Berlin

Institut für Sprache und Kommunikation
Fachgebiet Audiotechnologie

Fakultät I
Einsteinufer 17c
10587 Berlin
http://www.ak.tu-berlin.de

Thesis for M.S. in Audio Communication and Technology

Speech Emotion Recognition Using Convolutional Neural Networks

Gustav Sto. Tomas

[redacted]

09.05.2019

Supervised by:
Prof. Dr. Stefan Weinzierl
Athanasios Lykartsis

Technische Universität Berlin

Fakultät I
Geisteswissenschaften

Prüfungsausschüsse

Studiendekan und Vorsitzender

Prof. Dr. Friedrich Steinle

Eidesstattliche Erklärung

Ist jeder an der TU Berlin verfassten schriftlichen Arbeit eigenhändig unterzeichnet beizufügen!

Hiermit erkläre ich an Eides statt gegenüber der Fakultät I der Technischen Universität Berlin, dass die vorliegende, dieser Erklärung angefügte Arbeit selbstständig und nur unter Zuhilfenahme der im Literaturverzeichnis genannten Quellen und Hilfsmittel angefertigt wurde. Alle Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind kenntlich gemacht. Ich reiche die Arbeit erstmals als Prüfungsleistung ein. Ich versichere, dass diese Arbeit oder wesentliche Teile dieser Arbeit nicht bereits dem Leistungserwerb in einer anderen Lehrveranstaltung zugrunde lagen.

Titel der schriftlichen Arbeit

Speech Emotion Recognition Using Convolutional Neural Networks

VerfasserIn/VerfasserInnen* Name	Vorname	Matr.-Nr.
Sto. Tomas	Gustav	[redacted]

Betreuende/r DozentIn Name	Vorname	
Weinzierl	Stefan;	Lykartsis, Athanasios

Mit meiner Unterschrift bestätige ich, dass ich über fachübliche Zitierregeln unterrichtet worden bin und verstanden habe. Die im betroffenen Fachgebiet üblichen Zitiervorschriften sind eingehalten worden.

Eine Überprüfung der Arbeit auf Plagiate mithilfe elektronischer Hilfsmittel darf vorgenommen werden.

San Diego, 09.05.2019 [redacted]

Ort, Datum

*Bei Gruppenarbeiten sind die Unterschriften aller VerfasserInnen erforderlich.
**Durch die Unterschrift bürgen Sie für den vollumfänglichen Inhalt der Endversion dieser schriftlichen Arbeit.

Contents

Abstract

Speech emotion recognition is an upcoming subfield of automatic speech recognition that shares multiple similarities with mood recognition in music signals. Audio signals containing human speech are used as input to classification algorithms trained to recognize emotions in the form of audio features. This thesis outlines a study in which a data set of speech signals—containing semantically neutral recordings of professional actors portraying eight different emotions by altering their voices—is tested. Low-level descriptors (LLDs) were extracted as static subfeatures from the speech signals and classified with support vector machines (SVMs) and convolutional neural networks (CNNs). Furthermore, mel-spectrograms were extracted and fed into a CNN as 3D image vectors for classificaiton. The resulting accuracies from both the SVMs and CNNs for LLDs proved to be better than human raters from a previous study, with the CNN classifyer achieving the highest accuracy. The CNN for mel-spectrograms failed to achieve similar results, which is explained by lack of computational resources in the experiment setup.

1. Introduction

Speech emotion recognition is a field of research that combines elements of computer science, phonetics, music theory, and psychology. As a form of automatic speech processing, it is derived from basic digital signal processing methods such as the Fourier transform and convolutional filters. However, more advanced methods are heavily influenced by the neural computing and cognitive science that informs the deep learning algorithms for audio created by George E Hinton [1]. Beyond this, it is important to consider that complete measurable definitions do not exist of the psychological state referred to as human emotions in general and their relationship to the human voice in particular. Therefore, the expression of emotion through alterations of voice and intonation cannot either be said to have been fully explored or quantified. In music emotion recognition, sometimes referred to as mood classification, it is important to separate the notion of subjective emotions perceived by a listener when a particular piece of music is played back to them from the notion of musical mood. The latter notion shares many similarities with speech emotion recognition, and both research fields make use of similar audio features for classification tasks, where the goal is to automatically detect a portrayed emotion on the basis of some audio input into a classification algorithm and outputting one of a set of emotions or moods. This means that standardized features for general audio processing that have been used and tested both in music information retrieval and in automatic speech processing. Such features are used and tested in the present study as well, however the aim of the study was to compare these with newer processes of extracting features through deep neural networks.

This thesis outlines an experiment where a data set—containing audio recordings of professional actors portraying eight different emotional states by altering their voices—was used to train and test neural networks for audio classification. The data set was chosen on the basis of being publicly available and containing a larger set of files and range of emotion than other available data sets. A smaller data set with fewer recordings and fewer classes was further selected to serve as comparison.

The experiment was split into multiple steps, where the data was first processed to extract features and subfeatures. These were then used for classification with a support vector machine classifier and used as a baseline. The same feature set was subsequently used to train a convolutional neural network and classified by a fully connected neural network. Finally, a convolutional neural network was used to extract its own features from audio spectrograms—created from the original speech recordings—and a fully connected network was oncemore employed to classify these features.

2. Previous Studies

2.1. **Audio Processing and Classification Using Convolutional Neural Networks.** Deep neural networks have been applied to the task of audio classification, both in the form of recurrent neural networks (RNNs) and convolutional neural networks (CNNs). Böck & Schlüter [2] were influential in using spectrogram images as input for CNNs to detect musical onsets in audio signals. Similar approaches to using CNNs as feature extractors and predictors for speech signals have been taken for ASR [1], for speech synthesis [3], and for speaker and language detection [4]. RNNs combining audio and text data for emotion recognition were employed by [5], however the use of text data heavily influences the outcome, and the present study focuses solely on audio data.

2.2. **Speech Emotion Recognition.** Early research on the topic of emotional speech was largely focused on vocal prosody, with pitch as represented by fundamental frequency (F_0) envelope and range being the most thoroughly studied feature [7, 8, 9, 10]. It has been shown [7, 8, 9, 10] that the emotions anger and happiness are represented by large, near-identical values for F_0 envelope and range, with fearful speech having similar yet smaller values. On the other end of the spectrum, sadness displays the least amount of variety in F_0 envelope, and the smallest F_0 range. The emotions neutral and boredom display similar values for both range and envelope, yet are both somewhat more varied than sadness, whilst simultaneously being significantly less varied than neutral speech. These findings are somewhat reflected, yet less prominently, for temporal features [7, 8, 9, 10]. The emotions sadness and boredom are both depicted by slower speech rhythm than neutral speech, whilst fearful speech in turn displays the fastest speech rhythm.

More recent research has shifted the focus toward feature extraction of low-level descriptors (LLDs) from speech signals, and the classification thereof using methods of machine learning. The INTERSPEECH 2009 Emotion Challenge [11] was influential in standardizing a feature set for emotional speech, wherein LLDs and functionals were extracted and used for classification. As such, mean, maximum, minimum, range, standard deviation, etc, were calculated from extracted features such as root mean square (RMS) amplitude, zero-crossing rate (ZCR), and mel-frequency cepstral coefficients (MFCCs). Extracting these standard features from utterances in a data set consisting of emotional speech in combination with support vector machine (SVM) classifiers, Zhang, Essl, & Mower Provost [12] were able to accurately classify 80% of utterances; whereas human raters achieved only 62% accuracy on the same data set.

The application of DNNs for speech emotion recognition is quickly emerging, with recent papers employing RNNs, in particular Long Short-Term Memory-Networks (LSTMNs) [13] showing promising results. However, image processing based CNNs are currently gaining popularity for

emotion recognition, in particular as feature extractors used together with fully connected networks as classifiers [14, 15]. Thus far, however, features extracted by CNNs from spectrogram images have generally not been able to achieve the same accuracy levels as the standardized feature set from INTERSPEECH 2009 Emotion Challenge [11].

3. Methods

3.1. **Short-Term Fourier Transform.** When processing audio signals, blocks of hop size $\mathcal{H}$ are used to process overlapping frames of the signal, each with length $\mathcal{K}$ [16]:

$$i_s(n) = i_s(n-1) + \mathcal{H}$$

$$i_e(n) = i_s(n) + \mathcal{K} - 1$$

where $i_s(n)$ is the start sample and $i_e(n)$ is the stop sample of frame n. The reason for selecting shorter frames rather than continuous signals of arbitrary length comes down to the fact that the content of the audio signal may vary over time, and extracting information of value can become more difficult. Therefore, the FT is usually calculated over a signal frame $x(i)$ instead of an entire signal and is then referred to as the Discrete Fourier Transform (DFT) [16]:

$$X(j\Delta\Omega) = \mathfrak{F}\{x(i)\} = \sum_{i=0}^{K-1} x(i)e^{-jk9\Delta\Omega}$$

where $\Delta\Omega$ is the angular frequency difference between two frequency bins [16]:

$$\Delta\Omega = \frac{2\pi}{\mathcal{K} \cdot T_s} = \frac{2\pi \cdot fs}{\mathcal{K}}$$

The frame length $\mathcal{K}$ and sample rate fs can be used to calculate the frequency of a bin at index k using the formula [16]:

$$f(k) = \frac{\Delta\Omega}{2\pi}k = \frac{fs}{\mathcal{K}}k$$

Finally, a Short-Term Fourier Transform (STFT) is calculated [16]:

$$X(k,n) = \sum_{i=i_s(n)}^{i_e(n)} x(i)exp\Big(-jk \cdot (i - i_s(n))\frac{2\pi}{\mathcal{K}}\Big)$$

where $\mathcal{K} = i_e(n) - i_s(n) + 1$ is the length of the nth frame of a DFT. When plotting an SFTF along a time axis, it can be visualized as a *spectrogram* depicting the magnitude of each frequency component of the signal over time.

3.2. **Audio Feature Extraction.** Subfeatures in the form of low-level descriptors (LLDs) were extracted from audio vectors containing the speech recordings after first computing the STFT for each raw audio recording according to the formula above. Previous research and current state of the art in speech emotion recognition [12] have used the standardized LLD subfeature set OpenSMILE to extract audio features. In the present study, LLDs matching the feature set defined by OpenSMILE at the INTERSPEECH 2009 Emotion Challenge [11] were reproduced. After using predominantly the Python audio processing library librosa [17] to extract audio features in the form of time series data, LLDs and functionals thereof were whence calculated as static features. Thus, arithmetic mean, standard deviation, minima, maxima, and range values were extracted from standard features for speech signals: mel-frequency cepstral coefficients, fundamental frequency/pitch, root-mean-square amplitude, and zero-crossing rate. In addition, LLDs and functionals were computed from spectral features more commonly used in music information retrieval: centroid, bandwidth, rolloff, contrast, and flatness. Beyond this, rhythm features in the form of a spectral flux onset envelope with its local auto-correlation as well as estimated global tempo were considered and used to calculate LLDs.

3.2.1. *Low-Level Descriptors.* Sub-features in the form of static functionals of each audio feature are extracted and used for classification. The LLDs calculated are:

Maximum
Minimum
Range
Arithmetic Mean
Standard Deviation

Maxima and minima are found in each feature vector, and the range is calculated as $max(y) - min(y)$. The arithmetic mean is calculated as [16]:

$$\mu_x(n) = \frac{1}{\mathcal{K}} \sum_{i=i_s(n)}^{i_e(n)} x(i)$$

and the standard deviation is given as [16]:

$$\sigma_x^2(n) = \frac{1}{\mathcal{K}} \sum_{i=i_s(n)}^{i_e(n)} (x(i) - \mu_x(n))^2$$

The resulting feature set thus contained 94 feature vector for each audio recording, leading to a feature matrix $F = [1536, 94]$.

3.2.2. *Mel-Frequency Cepstrum Coefficients.* The mel scale was introduced by and Stevens & Volkmann [18] as a means to correlate acoustic frequency and perceived pitch in human psychoacoustics in a series of experiments. One step of the mel scale represents the change between what human subjects perceived to be two different tones, i.e. a change in mels equals a change in

pitch. The resulting non-linear mel scale can be modeled as such using the natural logarithm [16]:

$$\mathfrak{m}_F(f) = 1000 \cdot log_2 \left(1 + \frac{f}{1000\,Hz}\right)$$

It can equally be modeled using log base 10 [16]:

$$\mathfrak{m}_F(f) = 2595 \cdot log_{10} \left(1 + \frac{f}{700\,Hz}\right)$$

Mel-Frequency Cepstral Coefficients (MFCCs) are widely used in speech processing as a way to describe the filter of the filter-source theory, where the source is the vocal cords and the filter represents the vocal tract. The length and shape of the vocal tract are decisive for the output of speech sounds, and by calculating the cepstrum of an audio signal it is possible to separate the source from the filter. Furthermore, cepstral coeffecients display an uncorrelated variance, which stands in contrast to spectral coefficients, which in turn are correlated across separate frequency bands [19]. A cepstrum (a word formed by reversing the *spec* part of the word spectrum), then, is the spectrum of the log of the spectrum [19] and is computed by taking the log of each value of a power spectrum. Whereas a power spectrum of a signal depicts the frequency on the x axis and the amplitude on the y axis, a cepstrum is created by moving the x axis from the frequency domain into the time domain. The cepstrum is formally defined as the inverse Discrete Fourier Transform (DFT) of the log magnitude of the DFT of a signal $x(n)$ [19]:

$$c(n) = \sum_{n=0}^{N-1} log\left(\left|\sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}\right|\right) e^{j\frac{2\pi}{N}kn}$$

To calculate the MFCCs, a non-linear mel-warped spectrum was created by computing a series of 128 overlapping triangular filters. The frequencies of the input signal were mapped to 128 frequency bins corresponding to the amount of filters. The logarithm of each frequency bin was taken, and a Discrete Cosine Transform (DCT) was finally computed [16], similar to the DFT but using only real numbers. Along the input signal's time axis, 20 coefficients were calculated, thereby effectively creating a compact spectral envelope [16]:

$$v^j_{MFCC}(n) = \sum_{k'=1}^{K'} log\left(|X'(k', n)|\right) \cdot cos\Big(j \cdot \Big(k' - \frac{1}{2}\Big)\frac{\pi}{\mathcal{K}'}\Big)$$

where $|X'(k', n)|$ is the mel-warped spectrum at the time of the jth coefficient and the latter part of the formula describes the DCT [16].
Only the 13 first calculated MFCCs were saved and used, as these contained most useful information on the signal filter representing the vocal tract, while simultaneously being separated from any information about the signal source. Finally, subfeatures in the form of arithmetic mean, standard deviation, minimum, maximum, and range values were calculated for each of the 13 MFCCs.

3.2.3. *Fundamental Frequency and Pitch.* The fundamental frequency (f0) of a speech signal represents a speaker's pitch and correlates to the frequency with which the speaker's vocal folds generate sounds. One common implementation of a pitch tracking algorithm is to calculate the autocorrelation function (ACF) of a speech signal and then to locate the first real maximum in the ACF. The fundamental frequency is equal to the time period between 0 and the first maximum. Typical f0 values for speech signals are ca 120 Hz for male speakers, and ca 210 Hz

for female speakers.
A different algorithm for finding the f0 is implemented here based on quadratic interpolation by fitting a parabola to the nearest samples of a peak's neighbourhood consisting of three samples [17]. An interpolated peak location is given as a bin index:

$$p = \frac{1}{2}\frac{\alpha - \gamma}{\alpha - 2\beta + \gamma} \in [-1/2, 1/2]$$

where the sample indices are given as $\alpha = y(-1)$, $\beta = y(0)$, and $\gamma = y(+1)$. After detecting the peak location, its magnitude can be calculated thus:

$$y(p) = \beta - \frac{1}{4}(\alpha - \gamma)p$$

The calculation returns a vector of peak frequencies and magnitudes from which local maxima and minima are extracted as peaks and troughs. Global maximum and minimum are calculated from the resulting vector and in turn used to estimate pitch range and global mean f0.

3.2.4. *Chroma.* Chroma perception is a way describe the perceived notion of octaves, where frequency intervals of 2:1 are perceived to be similar [16]. Similar to the mel scale for human pitch perception, chromagrams are calculated by creating a set of 12 filter banks, each representing a pitch class within its octave. The filter banks are used to map the frequencies of a power spectrum to chroma bins and are subsequently normalized for each frame of an input power spectrum [17]:

$$\mathcal{C}(c, n) = \sum_{\{p \in [0,127] \,|\, p \bmod 12 = c\}} \mathcal{Y}(p, n)$$

for chroma bin $c \in [0, 11]$, and where p is a pitch as given by its corresponding MIDI identification number, and $\mathcal{Y}(p, n)$ is a log-frequency spectrum:

$$\mathcal{Y}(p, n) = \sum_{k \in P(p)} |X(k, n)|^2$$

3.2.5. *Root-Mean-Square Energy.* Sound intensity is measured as root-mean-square (RMS) amplitude for each frame of an input audio signal. It corresponds to the human perception of loudness and is an important feature in the classification of emotion in speech signals, with larger values corresponding to emotions such as anger and happyness, whereas smaller values indicate less enthusiastic speech such as neutral and sadness. The calculation of RMS is performed as [16]:

$$v_{RMS}(n) = \sqrt{\frac{1}{\mathcal{K}} \sum_{i=i_s(n)}^{i_e(n)} x(i)^2}$$

which returns a value between [0,1] per frame, where values closer to 0 indicate silence and values closer to 1 indicate loudness.

3.2.6. *Zero-Crossing Rate.* The amount of times a waveform passes 0, that is to say, the frequency with which it changes sign between - and + is known as the zero-crossing rate. In speech signals it commonly used both to detect voiced and unvoiced segments of a signal, and to calculate the fundamental frequency of the voiced segments. This is performed by directly correlating the amount of times the signal passes zero within a time frame to a signal's lowest pitch period. It can be assumed that longer periods of similar values across frames indicate periodic sounds, correlating to vowels in speech signals, and highly varied and very large values indicate white noise, correlating to unvoiced phonemes in speech signals. The formula is defined as [16]:

$$v_{ZC}(n) = \frac{1}{2 \cdot \mathcal{K}} \sum_{i=i_s(n)}^{i+e(n)} |sign[x(i)] - sign[x(i-1)]|$$

where *sign* is defined as:

$$sign[x(k)] = \begin{cases} 1, & \text{if } x(i) > 0 \\ 0, & \text{if } x(i) = 0 \\ -1, & \text{if } x(i) < 0 \end{cases}$$

The calculation returns a frame-wise value in the range $0 \leq v_{ZC}(n) \leq 1$.

3.2.7. *Spectral Shape and Timbre.* Timbre can be defined as the subjective aspect of a sound that cannot be reduced to pitch and/or loudness. That is to say, the timbre of a sound reveals some information about the sound source, regardless of the amplitude or frequency with which it is represented. For example, the timbre of the human voice is distinct from that of a trombone, even if both are presented at the same pitch and loudness. Formally, timbre can be explained as the relative amplitude and distribution of a sound's harmonics [16] and can be described and modeled with a set of spectral measures described below: spectral centroid, spectral bandwidth, spectral rolloff, spectral contrast, and spectral flatness.

3.2.8. *Spectral Centroid.* The center of gravity (COG) of a signal's spectral energy is referred to as its spectral centroid. The centroid—or mean—of the spectral energy describes an audio signal's "brightness," or "sharpness" [16]. It can be calculated in steps by first computing a magnitude spectrogram of a signal and then treating the spectrogram as a frequency-weighted sum thereof. Finally, each frequency-weighted frame of the spectrogram is normalized by its unweighted counterpart and the centroid is framewise extracted [4]:

$$v_{SC,m}(n) = \frac{\sum_{k=0}^{\mathcal{K}/2-1} k \cdot |X(k,n)|}{\sum_{k=0}^{\mathcal{N}/2-1} |X(k,n)|}$$

The calculation returns a frequency bin index in the range $0 \leq v_{SC,m}(n) \leq \mathcal{K}/2 - 1$, which was subsequently used to compute the centroid frequency f per bin index k using the formula [16]:

$$f(k) = \frac{\Delta\Omega}{2\pi} k = \frac{fs}{\mathcal{K}} k$$

where fs is the sample rate, and $\Delta\Omega$ is the angular frequency difference between two frequency bins [16]:

$$\Delta\Omega = \frac{2\pi}{\mathcal{K} \cdot T_s} = \frac{2\pi \cdot fs}{\mathcal{K}}$$

3.2.9. *Spectral Bandwidth.* A signal's instantaneous bandwidth can be described as the standard deviation of a magnitude spectrum around its spectral centroid. The result depicts the spectral spread of the signal from its centroid at a given frame, which for speech sounds can be understood as indicating the difference between transients in the form of consonants, and periodic sounds in the form of vowels. The formal definition is [16]:

$$v_{SB,m}(n) = \sqrt{\frac{\sum_{k=0}^{K/2-1}(k - v_{SC,m}(n))^2 \cdot |X(k,n)|^2}{\sum_{k=0}^{N/2-1}|X(k,n)|^2}}$$

where $v_{SC,m}$ is the spectral centroid calculated from the magnitude spectrum. The calculation returns a frequency bin index in the range $0 \leq v_{SB,m}(n) \leq \mathcal{K}/4$, which is subsequently used to compute the bandwidth around the centroid frequency per bin index.
A p-ordered bandwidth can equally be calculated as such [17]:

$$v_{SB} = \Big(\sum_{k} S(k)(f(k) - f_c)^p\Big)^{1/p}$$

where $S(k)$ is the magnitude of the spectrum at frequency bin k, $f(k)$ is the frequency at bin k, and f_c is the value of spectral centroid calculated from the magnitude spectrum. A p value of 2 returns a weighted standard deviation of the signal.

3.2.10. *Spectral Rolloff.* The maximum, minimum, and bandwidth frequency for each frame of a signal can be approximated by calculating the spectral rolloff. The rolloff can be described as a frame's center frequency for a spectrum bin [17], meaning that the center frequency contains the accumulated magnitudes of the STFT $X(k,n)$ [16] when these pass above a given threshold. The formula is given thus [16]:

$$v_{SR}(n) = i\Big|_{\sum_{k=0}^{i}|X(k,n)| = \mathcal{K}\cdot\sum_{k=0}^{K/2-1}|X(k,n)|}$$

where $\mathcal{K}$ is the threshold, which in the present study was set to 0.85. The calculation returns a frequency bin index in the range $0 \leq v_{SR}(n) \leq \mathcal{K}/2 - 1$, which was subsequently used to compute the center frequency per frame.

3.2.11. *Spectral Flatness.* A way to describe an input audio signal's tonal quality in terms of whether it contains more noise or periodic components is spectral flatness. It is calculated as "the ratio of geometric mean and arithmetic mean of the magnitude spectrum" [16]:

$$v_{Tf}(n) = \frac{\sqrt[K/2]{\Pi_{k=0}^{K/2-1}|X(k,n)|}}{2/\mathcal{K}\cdot\sum_{k=0}^{K/2-1}|X(k,n)|} = \frac{exp\Big(2/\mathcal{K}\cdot\sum_{k=0}^{K/2-1}log(|X(k,n)|)\Big)}{2/\mathcal{K}\cdot\sum_{k=0}^{K/2-1}|X(k,n)|}$$

where the result is a value between [0,1] for each frame of an input magnitude spectrum. Values closer to 0 indicate tonal spectra and larger values closer to 1 indicate a noisier spectrum.

3.2.12. *Spectral Contrast.* Octave-based spectral contrast is calculated by computing spectral peaks and troughs for six frequency sub-bands for each frame. By doing so, the spectral contrast reflects both the relative spectral characteristics and the distribution of harmonic and non-harmonic components of for each frequency sub-band. the algorithm is based on finding local maxima and minima in a neighbourhood by calculating whether a value in a frame is larger or smaller than other values in its neighborhood by a given α value. The contrast, then, is the difference between a peak and a trough [20]:

$$SC_k = Peak_k - Trough_k$$

where $Peak_k$ and $Trough_k$ are calculated, respectively:

$$Peak_k = log\{\frac{1}{\alpha N}\}\sum_{i}^{\alpha N} = 1X'_{k,i}$$

$$Trough_k = log\{\frac{1}{\alpha N}\}\sum_{i}^{\alpha N} = 1X'_{k,N-i+1}$$

where N is the total number of the kth sub-band in the set $k \in [1, 6]$ and X' is an FFT vector of the kth sub-band sorted in descending order: $\{X'_{k,1} > X'_{k,2} >, ..., > X'_{k,N}\}$. The value α was set to 0.02.

3.2.13. *Rhythmic Features.* Early research in speech emotion recognition used speech rate and/or speaker rate as measurements of speaker tempo. These measurements are based on the calculation of spoken syllables in a transcription of an utterance divided by the duration of the signal. The present study does not make use of transcriptions and instead works solely with the actual audio signals, and thus instead implements onset envelopes and beat histograms to measure the tempo of a speech signal. Periodicities in the signal's amplitude envelope are detecting and assumed to reflect syllabilic structure of the speech signal as represented by vowels and consonants.

3.2.14. *Spectral Flux Onset Envelope.* Onsets of spectral information can be calculated as novelty functions that indicate the rate of spectral change in an audio signal, a notion that some new information is presented. Onset detection can thus be understood as the slope of an audio signal's envelope computed over its peaks. A spectral flux onset envelope with local maximum filtering is calculated according to the formula given by [22]:

$$SF^*(n) = \sum_{m=1}^{m=M} H\big(X_{log,filt}(n,m) - X^{max}_{log,filt}(n-\mu m)\big)$$

where μ is a frame offset parameter and the maximum filtered spectrogram is:

$$X_{log,filt}(n,m) = max(X_{log,filt}(n, m-1:m+1))$$

and the log filtered spectrogram is:

$$X_{log,filt}(n,m) = log_{10}(|X(k,n)| \cdot (F(k,m)+1)$$

where m is a bin index of a scaled frequency and F being a triangular filter from a filterbank used to calculate the scale. In the present study, the mel scale was used to map the input signal to 128 mel bands [17]. The calculation returns a vector of values for onset strength over time representing the onset envelope.

3.2.15. *Autocorrelation Beat Histogram.* Once a novelty function has been computed, the onset envelope can be used to calculate an autocorrelation function. The ACF detects a signal's similarity with itself and can be used to detect a signal's beat histogram by detecting peaks representing beat periods in a local autocorrelation. The ACF is calculated as [22]:

$$\mathcal{T}^A(t,\tau) = \mathcal{A}(t,l)$$

for each BPM tempo $\tau = 60/(r \cdot l), l \in [1:N]$, where r is a step size and l is a time lag of the ACF. $\mathcal{A}(t,l)$ is the ACF as defined by [22]:

$$\mathcal{A}(t,l) = \frac{\sum_{n\in\mathbb{Z}} \Delta(n)\Delta(n+l) \cdot W(n,t)}{2N+1-l}$$

for time $t \in \mathbb{Z}$ and time lag $l \in [0:N]$, and where W is a windowing function. The calculation returns a localized autocorrelation that is used to find a maximum value which in turn is used to estimate the global tempo in beats per minute (BPM).

4. Classification: Support Vector Machines

4.1. **Hyperplanes.** Support Vector Machines (SVMs) are a group of classifiers based on separating data points in a p-dimensional space with a $p-1$-dimensional hyperplane and assigning a class to each data point, depending on which side of the hyperplane it is located. A hyperplane is defined as [23]:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0$$

where β_i are model parameters, such that any point for which the equation holds true is a point on the hyperplane. With input data $X = (x_1, x_2, ..., x_p)$, it follows then, that if either

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} > 0$$

or

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} < 0$$

is true, then the data point is located on either side of the hyperplane. What this in turn means, is that in the case of a binary classification task, the two classes $y_1, ..., y_n \in [-1, 1]$, are assigned as:

$$y_i = 1 \quad \text{if} \quad \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} > 0$$

and

$$y_i = -1 \text{ if } \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} < 0$$

Depending on the nature of the data, the simultaneous existence of an infinite number of hyperplanes is possible. As discussed further below, support vector machines are used to find the hyperplane that maximizes the distance between data points and the hyperplane.

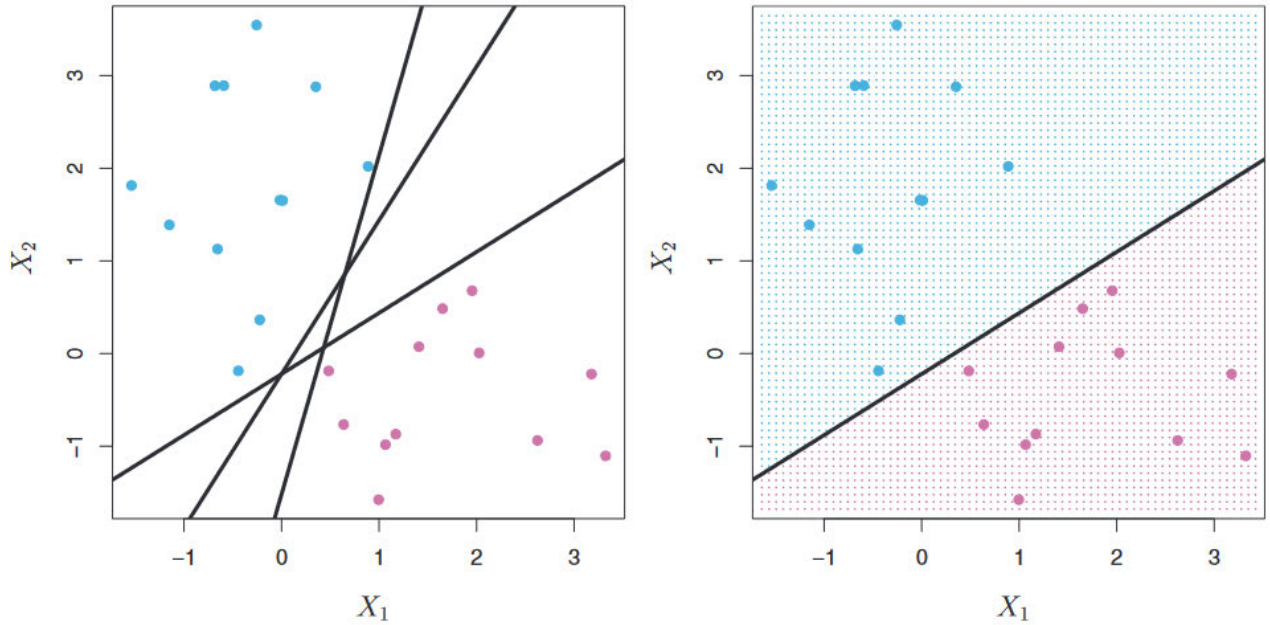


Illustration from [23]: Left: Three possible hyperplanes separating blue and red data points. Right: A hyperplane illustrating two subspaces, where data points in the blue space are assigned one class, and data points in the red space are assigned another class.

4.2. **Maximal Margin Classifier.** The distance between a data point and a hyperplane is known as the *margin*. Out of infinitely many possible hyperplanes, the best fit model is the one that maximizes the margin, so that the hyperplane is located with equal margin on each side to two data points of different classes that lie the closest to each other in the data space. This can be formalized as [23]:

$$\text{maximize } M_{\beta_0,\beta_1,...,\beta_p}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_1(\beta_0 + \beta_1 xi1 + \beta_2 x_{i2} + ... + \beta_p xip) \geq M \, \forall \, i = 1, ..., n$$

where M is the margin given the coefficients $\beta_0, \beta_1, ..., \beta_p$. The constraints are given so that each data point is assigned the correct class depending on the side of the hyperplane it is located and that its distance to the hyperplane is larger than the value given by M. The data points that maximize the margin are known as *support vectors* because moving them in either direction of the feature space will equally move the separating hyperplane.

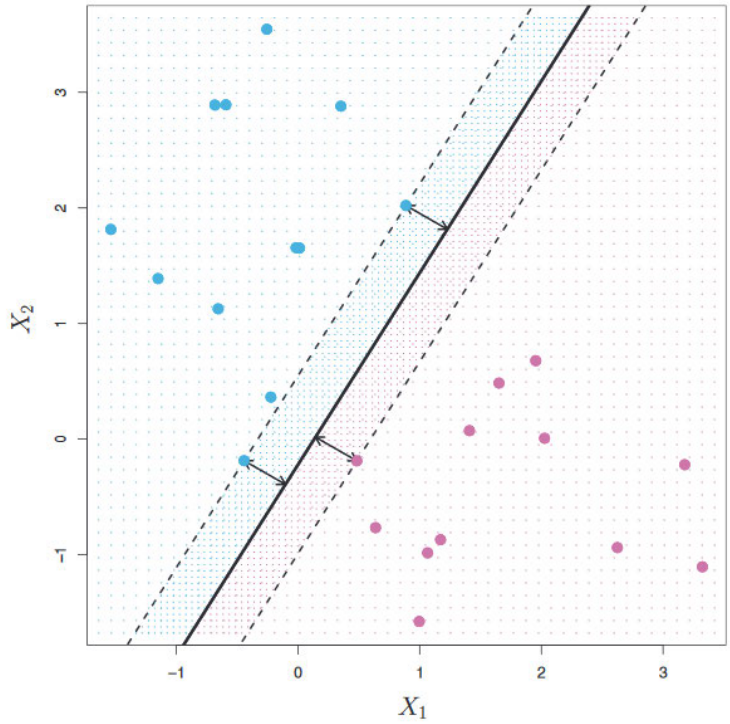


Illustration from [23]: A hyperplane with the data points that maximize the margin. Dotted lines indicate support vectors.

4.3. **Support Vector Machine Classifier.** Since real data rarely is as easily classifiable as implied above, SVMs differ from maximal margin classifiers in that they implement a *kernel* function to account for non-linear classification. Rather than classifying the input samples themselves, SVMs calculate the inner product of the input [23]):

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

where K is a linear kernel function. By calculating the inner product similarity between two input values, a class can be assigned to each data point by taking the entire data set into account without being computationally expensive. The SVM decision function for a test point x can be formalized as [24]:

$$h(x) = sign\Big(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + \rho\Big)$$

where $\alpha_i > 0$ is a regularizing weight and ρ is a bias term that both are learned for each data point. By setting $\alpha_i = 0$ for data points that are considered further away from the hyperplane, the algorithm reduces the amount of data needed for classification.

4.4. **Multiclass SVMs: One vs One.** For non-binary classification tasks, as in the present study with $K = 8$ classes, multiple SMVs are constructed and classes are compared pair-wise. That is to say, $K \cdot (K-1)\,/\,2$ classifiers are constructed and each one trains on and classifies data from two classes. This way, $\binom{K}{2}$ classifiers are trained and every data point is classified by

each of them. Every data point is finally assigned to the class to which it was most frequently assigned during the eight pair-wise classifications [23].

5. Feed-Forward Neural Networks

Artificial neural networks are, in their most basic form, nested functions that consist of layers, where each layer performs some function and feeds its output to the next layer of the network architecture. A single layer network would thus be:

$$y = f_{NN}(x)$$

and a 3-layered nested network would be [25]:

$$y = f_{NN}(x) = f_3(f_2(f_1(x)))$$

where f_1 and f_2 are functions. Each network layer performs some function that can easiest be explained through matrix multiplication, such as $X = YZ$ with dimensions l, m for Y and m, n for Z [26]:

$$x_{i,j} = \sum_{k=0}^{k=m} y_{i,k} z_{k,j}$$

resulting in a matrix of dimension l, n. Thus, a neural network function can be described as matrix multiplication of weights W, input X, and the addition of a bias term B [26]:

$$L = XW + B$$

where L is the output of the layer, and W and B are learned during training through a process known as *stochasticgradientdescent*. Knowing this, the function $y = f_{NN}(x)$ can be rewritten as [25]:

$$f_1(z) := g_l(W_l z + b_l)$$

where l is the index of the layer and g is an activation function. When using feef-forward neural networks as classifiers, a probability distribution is given over all possible classes for an observation, and a loss function is calculated to evaluate the fit of the model [26]:

$$Pr(A(x)) = \sigma(xW + b)$$
$$L(x) = -log(Pr(A(x) = a))$$

where σ is a softmax classifier and L is the loss function. For detailed descriptions on each of these functions, see further below.

6. Convolutional Neural Networks

Convolutional Neural Networks, often abbreviated as CNNs, or sometimes ConvNets, are a special kind of feed-forward neural network, where the initial, and often multiple subsequent, layers consist of a convolutional layer. A convolution, in the DSP sense, is merely the combination of two signals to create a third. The simplest form of a convolution by that definition is a δ function x convolved with an impulse response h, resulting in an output y:

$$y = x * h$$

When describing CNNs, the δ function is referred to as *input* and the impulse response is referred to as a $filter\ kernel$ or simply $kernel$. The output is called a $feature\ map$ and contains information about the input, similar to the way that the output signal from a linear combination of a δ function and an impulse response reveals information about the input signal and vice versa. In brief terms, CNNs make use of a neural network architecture known as convolution and pooling, which recognizes and extracts predicative patterns from larger structures and produces new vectors containing these features. Thus, locally informative sub-feature vectors are extracted from input audio feature arrays and used to match similar sub-feature vectors. Each layer of the network extracts features from the previous layer and inputs these into the next according to some function assigning weights to input features and thereby producing model parameters. Thus, just like the aforementioned impulse response, a signal x at time t can be detected and estimated in a new function s using a convolution operation applying weights w to each x at t [27]:

$$s(t) = (x * w)(t)$$

The kernel can be thought of a window that slides across the input signal in increments and performs some function outputting the feature map while learning weights and bias terms in such a way that the function best describes the input. The input signal may be multidimensional and can therefore be convolved over multiple axes, which combined with an equally multidimensional kernel K (here with a 2D input array I) can be described as below [27]):

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

For example, imagine a two-dimensional input signal in the form of a 3 x 3 matrix, P, and an equally shaped filter kernel, F [25]:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad F = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$

By performing a convolution operation on the two matrices P and F, an output matrix containing the dot products is computed. As the input matrix only contains positive values on positions p_{12}, $p_{21,22,23}$, and p_{32} and all other values of P are 0, the output matrix will reflect this shape; matrices with positive values located at identical indices will result in larger output values:

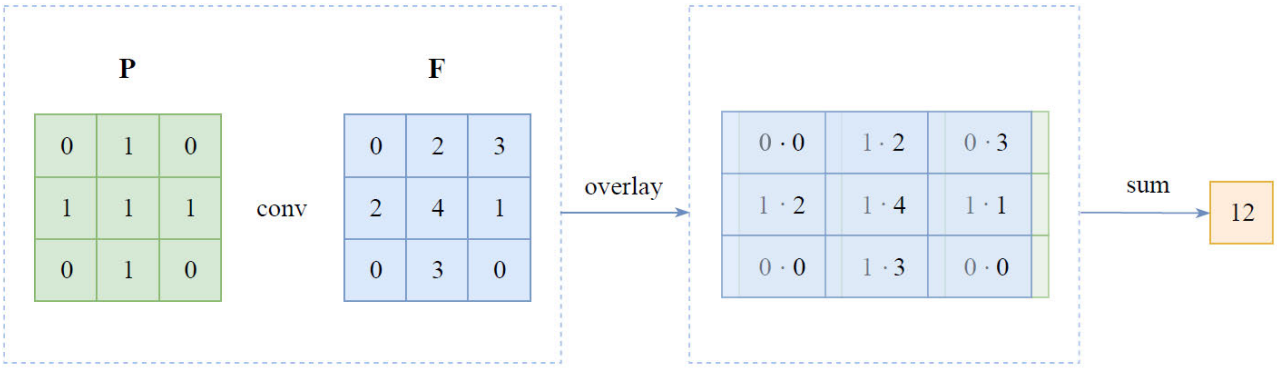


Illustration from [25]: Convolution of input and filter.

It is worth noting, that CNNs historically were developed to mimic visual perception in humans and thus were designed with the intention of being used for image processing. Is is therefore easy to explain their operations by describing such a use case.

6.1. **Filter Kernel.** Every value of an input matrix to a CNN can be thought of as a pixel in an image. Grayscale images contain values between 0, for black, and 255, for white, whereas color images contain values between 0 and 255 in three separate dimensions, or channels, representing the colors red, green, and blue. The size of the matrix corresponds to the width and height dimensions of the image.
Each convolutional layer in the network architecture contains multiple filters, each of which consists of a kernel that slides across the pixels of the image at a step size, or *stride*, that is set beforehand. At every step, the filter performs some function on the input image and outputs the resulting values into the feature map. For each iteration of the network function, weights and bias terms are learned by the system to optimize the output values. A sliding kernel with a size of 2 x 2 and a stride of 2, meaning that each computation is calculated on adjoined but not intersecting neighborhoods of the input matrix, performing four convolutions on a 4 x 4 input matrix and outputting the results is depicted below:

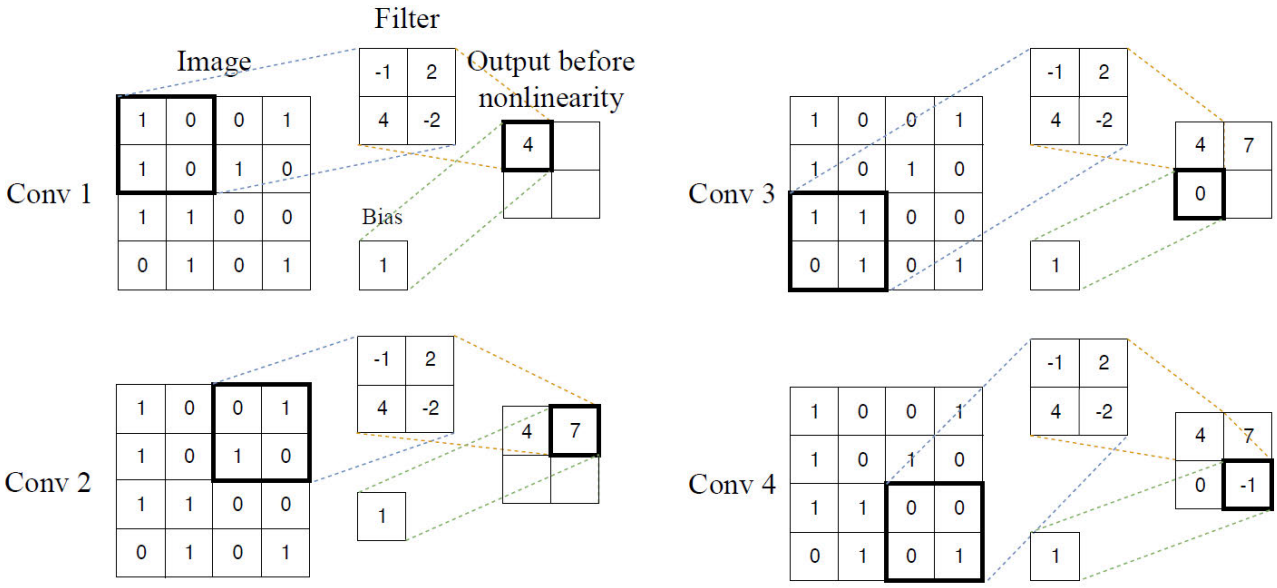


Illustration from [25: Filter kernel sliding across image matrix.

However, as the spectrograms contained in the dataset used for the study are color images with 3-dimensional RGB values, each convolution is performed thrice by the same filter kernel, once for each color channel. The combination of all three channels is referred to as a *volume*, and the output value in the illustrated example below is calculated as the sum of the dot product of all channels in the volume and the filter kernel, and finally adding the bias term [25]:

$$(-2\cdot 3+3\cdot 1+5\cdot 4+1\cdot 1)+(-2\cdot 2+3\cdot(-1)+5\cdot(-3)+-1\cdot 1)+(-2\cdot 1+3\cdot(-1)+5\cdot 2+-1\cdot(-1))+(-2)$$

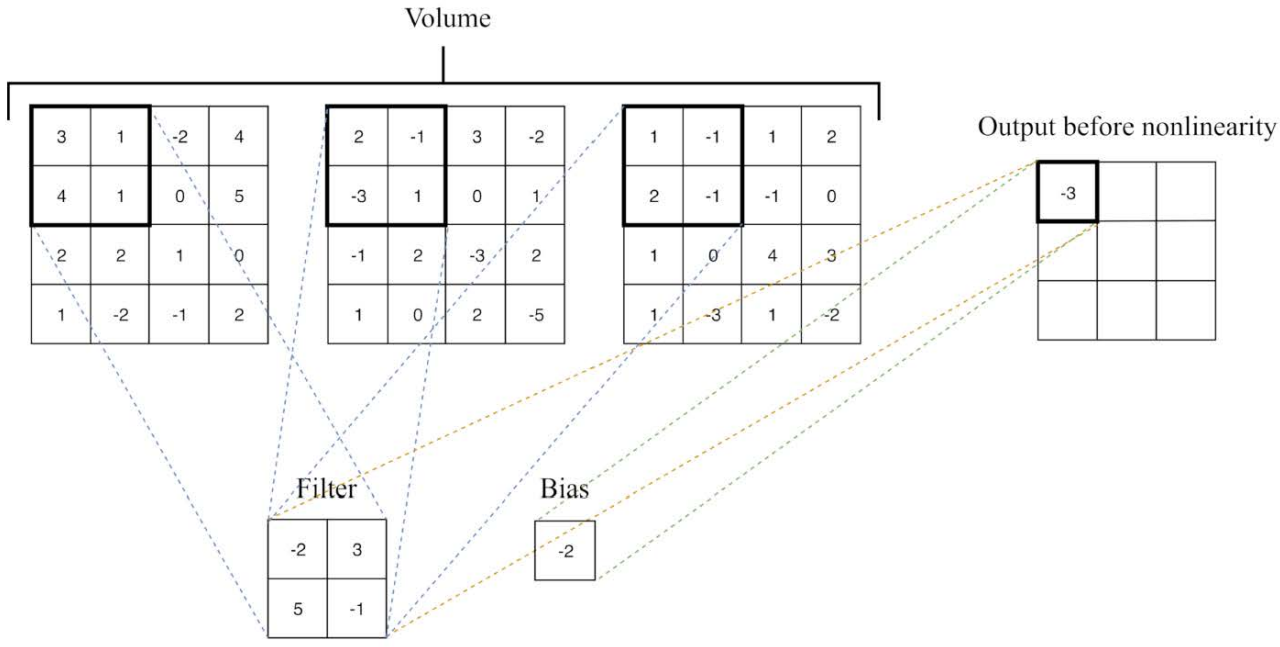


Illustration from Burkov (2019): Convolution of 3D RGB volume.

One important aspect of convolution is padding, which allows the filter to slide across the values closest to the boundaries of the image matrix, even if the size of the kernel is larger than the remaining amount of input pixels. It is typical to pad the image with zeros, as these do not interfere with the computation of the output feature map, yet still allow the convolution to make use of the entire input matrix, regardless of stride value.

6.2. **Non-linear Activation Function.** The description above of the input P and the filter F could be formalized as a simple linear function:

$$y = XW$$

where X is the input and W the weights assigned by the filter. However, if this was the only property of a neural network, then adding more layers in a deeper architecture would not result in anything that could not already be captured by the first layer. This property can be described with linear units U being fed into a network layer V for matrix multiplication [26]:

$$y = (xU)V$$

$$= x(UV)$$

It follows then, that with the product of U and V being W: $W = UV$, adding further layers to the system does not introduce more information about the input than what is already captured by $y = XW$. For this reason, a non-linear activation function is introduced between each convolution layer. Frequently used, including in this study, is the rectified linear unit ($ReLU$), which picks the max value in an input argument and sets negative values to 0:

$$\rho(x) = max(x, 0)$$

where ρ is the activation function ReLU. An important property of ReLU is that the output range of the function goes from 0 to ∞. This means that finding the gradient of the output value, which is effectively how the network calculates its loss function and updates its weights, is easier than with a limited range of output values. This solves a common problem for network architectures without non-linear activation functions, where the gradient can reach values approximating zero, causing the gradient to virtually explode or vanish [26].

6.3. **Pooling.** As mentioned further above, CNNs make use of an architecture referred to as *convolution and pooling*. Together with the convolution layer containing the filter kernel and the activation function, the other aspect of the architecture is a pooling layer. In comparison to the convolution layer, the pooling layer does not produce any model parameters, but instead reduces them. However, pooling still provides information about the feature map. This procedure is performed by once more letting a window slide across the image, similar to the filter kernel. For each window, or *patch*, some statistics are provided, such as either local max or average value, or global max or average. Depicted below is local max pooling:

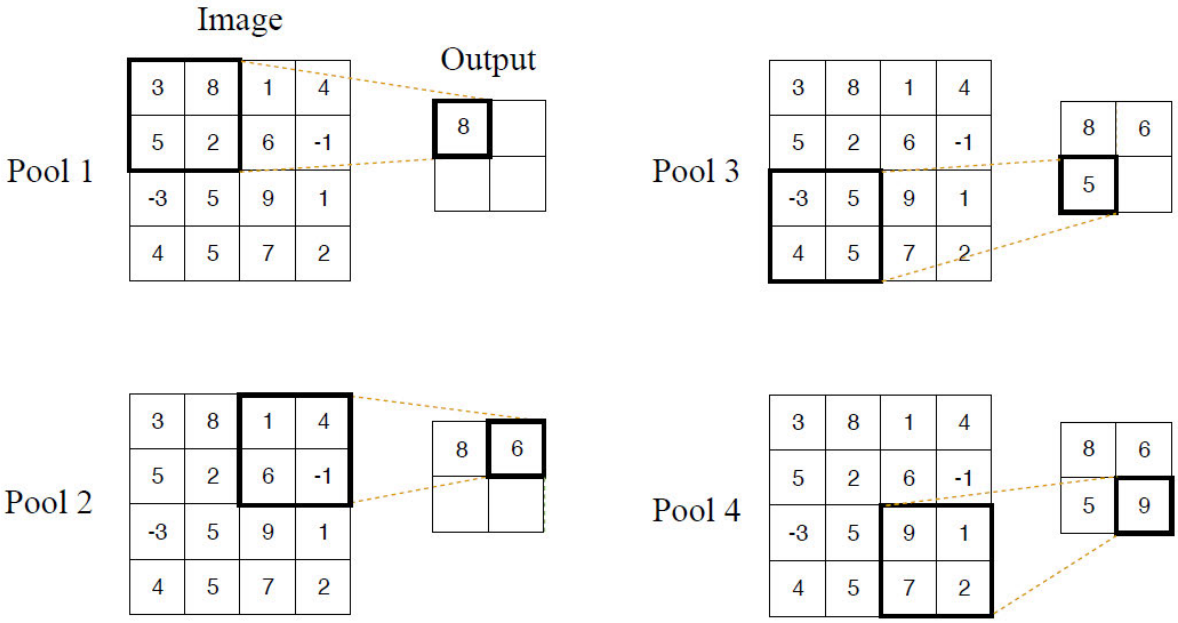


Illustration from [25]: 2D Max pooling.

Pooling dramatically reduces the amount of parameters in the network by providing only statistics on the input and outputting these into the next layer. This simultaneously serves to enhance the speed of the network.

6.4. **Loss Function: Cross-Entropy.** While training a neural network to recognize any data, it is necessary to be able to detect how well the network is performing and to what degree the weights need to be updated in each iteration. *Cross-entropy loss* is a function that estimates the negative of a cross-entropy, which in turn is a function to measure how similar two distributions are [25]. When used as a loss function, it defines how incorrect a model prediction is when classifying input features. Consider the following cross-entropy loss function X (Charniak, [25]:

$$X(\Phi, x) = -\ln\, p_{\Phi}(a_x)$$

where Φ is a model parameter, x is a feature, and p is a value from a probability distribution that sums up to 1 and refers to a class a. In the present study, 8 classes were present in the dataset, meaning that the probability for each class is $\frac{1}{8}$ and the classification is performed by a softmax function (see further below) assigning probabilities to each class to determine how likely they are to best explain the input. The most likely class receives the largest value in the probability distribution, e.g. 0.7. The loss function, then, is the negative log probability of that value (0.3), such that the larger the class probability value, the smaller the loss value and vice versa. The log value is used, because the natural logarithmic function $\ln x$ increases rapidly between values 0 and 1, precisely the values in the probability distribution [25].
The network outputs feature vectors containing real numbers, so in order to get probabilities for classification and subsequently the calculation of the loss function, a softmax function is utilized [25]:

$$\sigma(x)_j = \frac{e^{x_j}}{\Sigma_i e^{x_i}}$$

where σ is the softmax and x the input value. Since the exponential value of x is taken, it will always be a positive number, divided by the sum of all possible values of x, which, again, is $\frac{1}{8}$, resulting in probabilities for input C belonging to class c for $c \in [0, 1, ..., 7]$ [25]. Thus, a probability distribution is assigned by turning the input vector containing pixel values, into values between 0 and 1 for use in the loss function. The input values to the softmax function are referred to as *logits*.

6.5. **Stochastic Gradient Descent.** The computation of the loss function serves to update the weights w_j and bias b_j for each unit in the network iteratively and is referred to as *back propagation*. To update the values of the weights, the gradient of the loss function is calculated, which gives the method its name: *stochastic gradient descent*, or *SGD*. The stochastic part of the name refers to the procedure of setting a value called *batch size* that decides how many gradients should be calculated before updating the parameter values, as updating after each single calculation would be computationally expensive. To further reduce the computational power needed, rather than to search for the global minimum of the gradient, SGD finds a local minimum of the loss function and uses this value as an approximation.

6.5.1. *Updating the bias parameters.* Logits l_j can be described as a linear function: the dot product of input and weight plus bias [25]:

$$l_j = b_j + \mathbf{x} \cdot \mathbf{w_j}$$

This simply indicates that a change in bias or weights will be reflected by the logit. It follows then, that an altered value of l_j, which is used to calculate the probability values, subsequently alters the cross-entropy, which in turn is used to update the values in the next iteration. This can be formalized by looking in turn at the gradient of the loss function with respect to the bias term, the logit, and the weights. Below is the formula for the gradient with respect to the bias term b_j [25]:

$$\frac{\partial X(\Phi)}{\partial b_j} = \frac{\partial l_j}{\partial b_j} \frac{\partial X(\Phi)}{\partial l_j}$$

However, the first partial derivative can also be written out as such:

$$\frac{\partial l_j}{\partial b_j} = \frac{\partial}{\partial b_j}(b_j + \sum_i x_i w_{i,j}) = 1$$

where $w_{i,j}$ is the ith weight of jth linear unit [25]. This goes to say, that the derivative of the loss function with respect to b_j is 1, as a change in bias plus the linear function described above only affects the bias value itself.
The derivative with respect to the logit l_j is slightly different. As explained further above, the logit is used to calculate the probability p of belonging to a class a by the softmax function. Thus, the only term that is affected by the probability is X, with l_j causing the change in probability [25]:

$$\frac{\partial X(\Phi)}{\partial l_j} = \frac{\partial p_a}{\partial l_j} \frac{\partial X(\phi)}{\partial p_a}$$

This can be further developed as a change in probability with respect to the logit:

$$\frac{\partial X(\phi)}{\partial p_a} = \frac{\partial p_a}{\partial l_j}(-\ln p_a) = -\frac{1}{p_a}$$

where the probability of the jth value belonging to class a is assigned by the softmax function σ:

$$\frac{\partial p_a}{\partial l_j} = \frac{\partial \sigma_a(l)}{\partial l_j} = \begin{cases} (1-p_j)p_a & a = j \\ -p_j p_a & a \neq j \end{cases}$$

Combining the equations above to find the derivatives with respect to bias and logits, the formula for updating the bias terms of the network is [25]:

$$\Delta b_j = \mathcal{L} \begin{cases} (1-p_j) & a = j \\ -p_j & a \neq j \end{cases}$$

where $\mathcal{L}$ represents the hyperparameter *learning rate*, which regulates the value by which the parameters will be updated with each iteration.

6.5.2. *Updating the weight parameters.* Once the derivatives have been found for logit and bias, calculating the cross-entropy as a function of the weights is not far. The equation for finding the partial derivatives with respect to the weights, is given as such [25]:

$$\frac{\partial X(\Phi)}{\partial w_{i,j}} = \frac{\partial l_j}{\partial w_{i,j}} \frac{\partial X(\Phi)}{\partial l_j}$$

where the second partial is equal to that of the bias term's described above and can be reused. The first partial derivative can be written out as [25]:

$$\frac{\partial X(\Phi)}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}}(b_j + (w_{1,j}x_1 + ... + w_{i,j}x_i + ...)) = x_i$$

Finally, the formula for updating the weight parameters is defined as [25]:

$$\Delta w_{i,j} = -\mathcal{L} x_i \frac{\partial X(\Phi)}{\partial l_j}$$

where the minus sign is necessary to *decrease* the loss as a function of the updated parameter.

6.6. **Measures against Overfitting.** A trade-off exists between model complexity and explainability, but also between model complexity and model bias: The more parameters a model has, the more likely it is fit its parameters perfectly to the training data. With CNNs being located on the farther side of the complexity spectrum, it is not unusual for a network to have millions of trainable parameters, which easily can lead to overfitting, especially on a smaller dataset like the one used for this particular study. The following measures were undertake in the present study: regularization, dropout, batch normalization, and data augmentation. Each are described below.

6.6.1. *Regularization.* In traditional machine learning, the use of L1 and L2 regularization serves to control against overfitting. Regularization can easiest be explained as introducing a cost function for values that rise too quickly, controlled by a hyperparameter λ that is multiplied by the L1 or L2 term. A larger λ value will shrink the input coefficients increasingly closer to zero, whereas a λ value of 0 will perform no regulation at all. Shrinking the coefficients will reduce variance between training set and test set, as values become more aligned with each other [23]. The difference between L1 and L2 regularization is simply that L1 applies the cost function on the sum of all input coefficients, whereas L2 applies to the sum of squares. Shrinking the L1 features will result in some features being set to 0, a form of penalization that effectively works as feature selection [23]. L2 regularization, on the other hand, is computationally efficient and performs well as a means against overfitting in many tasks, but does not inherently perform any feature selection.
In the study at hand, regularization was used for penalizing the activity function in the first classification network.

6.6.2. *Dropout.* One of the most commonly employed strategies against overfitting is to leave out a set amount of units in each iteration. This simply goes to say that the network does not look identical every time the data is fed through it. A hyperparameter with a value of 0.5 means that for each batch of input matrices, the network assigns a probability of 0.5 to each nonoutput unit in the architecture [27]. By doing so, there is a 50% chance for each network unit to be included in each iteration. In the end, this means that the units are forced to see some new data in each iteration, rather than simply memorize the data, which would lead to overfitting. However, using a larger value for the dropout hyperparameter, or adding multiple dropout layers to the architecture can lead to slower training time for the network, or even underfitting.

6.6.3. *Batch Normalization.* It has been shown that not only does training converge faster, but also that the need for dropout is reduced by the introduction of *batch normalization*. The input data is *whitened*, that is to say, the data of the input layer is linearly transformed to have zero mean and unit variance, similar to *Principal Component Analysis* [28]). Since each network layer uses the output of the previous layer as its input, whitening can be introduced at multiple layers in the network. Indeed, adding batch normalization together with each activation function results in a normalized input at every iteration. Furthermore, batch normalization regularizes parameter values and aids in controlling against vanishing gradients. The algorithm introduced by [28] effectively whitens each input batch to have zero mean and variance of 1. Applying normalization to mini batches rather than the entire data set leads to reduced training time and higher accuracy in comparison to multiple dropout layers. Consider x as input u + bias b, and $\mathcal{X} = (x_1, ..., x_N)$ [28]:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}}$$

where $E[x] = \frac{1}{N}\sum_{i=1}^{N} x_i$.

For a mini batch of input data $(x_1, ..., x_m)$ being computed with stochastic gradient descent, an estimate of mean and variance for the data set is estimated at each iteration. This means that at each parameter value update, statistics are provided for normalization. Thus, the linear transformation BN gives two new learnable parameters, γ and β, that are updated together

with the weights [28]:

$$BN_{\gamma\beta} : x_1, ..., x_m \rightarrow y_1, ...y_m$$

where y_i are the linearly transformed input values. This way, any input x will now instead be whitened as $BN(x)$. In the very same way, a convolution layer, with its ReLu activation function g, that can be read as $z = g(Wu+b)$ will after batch normalization instead be written as [28]:

$$z = g(BN(Wu))$$

where the bias term b is replaced by the BN parameter β, since normalization entails subtraction of the mean, effectively cancelling the addition of bias. For each convolution layer's output feature map, BN applies the linear transformation on the input batch, such that both the values and spatial locations in the feature map are considered. This way, for each feature map, the γ and β parameters are learned and the input values are linearly transformed [28].

6.6.4. *Data Augmentation.* Data augmentation is an important step of preprocessing the dataset. It makes the dataset robuster for generalization to unseen data by altering the original dataset according to some function. In image processing, it is common to e.g. scale, flip by an axis, or alter the coloration of the data. For every augmentation operation, the dataset increases in volume by a factor of 2, thus enhancing the size of the dataset, which in and of itself is useful—in particular when using a smaller dataset, as is the case in the present study.

After converting the audio waveforms to mel-spectrograms, each mel-filter bank is processed to randomly warp the frequency axis of the mel-spectrogram, thereby simulating differences in speaker vocal-tract length. Traditionally, Vocal Tract Length Normalization (VTLN) has been used to normalize speech data, whereas Vocal Tract Length Perturbation (VTLP) achieves the opposite by generating new data, in which an input mel-spectrogram is used to output a frequency-warped version thereof, effectively imitating a different speaker with a longer or short vocal tract [29].
The formula for linear scaling VTLN is given thus, with α values given by [29] as ranging between 0.9 and 1.1 for VTLP, instead of VTLN values between 0.8 and 1.2:

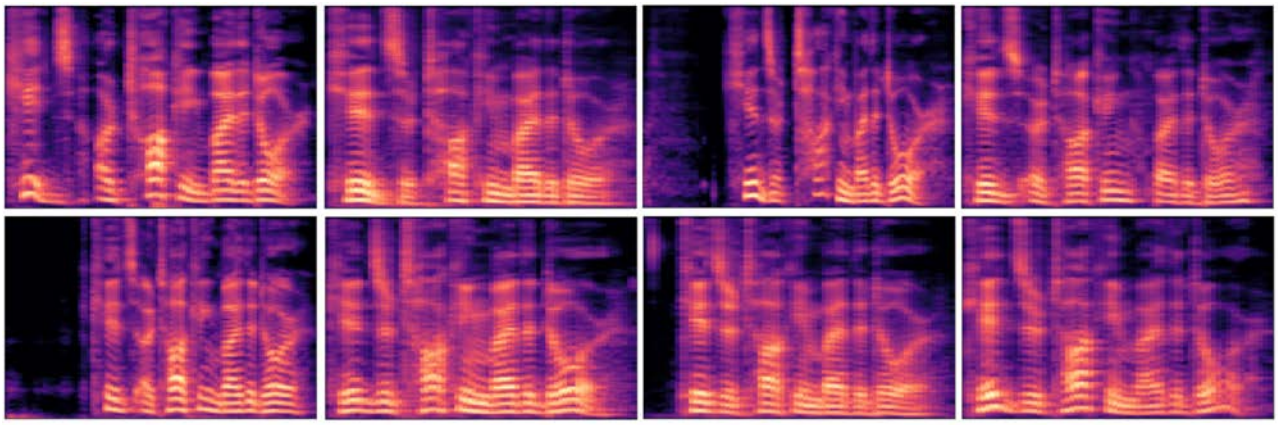
$$G(f) = \begin{cases} \alpha f, & 0 \leq f \leq f_0 \\ \frac{f_{max} - \alpha f_0}{f_{max} - f_0}(f - f_0) + \alpha f_0, & f_0 \leq f \leq f_{max} \end{cases}$$

The α values were uniformly randomized within the given range to create an augmented training set, twice as large as the original dataset, whereas the test set contained only spectrograms without perturbation.

7. Data Sets

7.0.1. *RAVDESS.* The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) described in Livingstone & Russo (2018) [30] was used as dataset for the project. The data set contains 1440 recordings of semantically neutral utterances that were produced in Toronto, Canada using 24 English speaking actors (whereof 12 are male and 12 are female). Two different sentences were spoken: "Kids are talking by the door" and "Dogs are sitting by the door." Each of the sentences contain seven syllables. Eight separate emotional states are represented: neutral, calm, happy, sad, angry, fearful, disgust, and surprised. The data set was validated by

319 human raters, achieving 62% average accuracy for the data set. The average length of each audio recording was ca 3 seconds.



Mel-spectrograms calculated from the RAVDESS dataset. Each image depicts one emotion from the same sentence and the same speaker. Top row, left to right: angry, calm, disgust, sad; bottom row, left to right: fearful, happy, neutral, surprised.

7.0.2. *Berlin EmoDB.* The Berlin Database of Emotional Speech [31], developed at TU Berlin will be tested as well. The data set consists of 535 utterances from 10 German speakers (whereof five are male and five are female). Seven separate emotional states are represented: neutral, bored, happy, sad, angry, fearful, and disgust. The data set was validated by 20 human raters, achieving 80% average accuracy for emotional domain. However, due to the limited size of the database it was only sparsely tested in the present study in order to verify the generalization of one model from one data set to another.

8. Procedure

The experimental design for recognizing emotions in speech signals was set up in several separate steps. Largely, the experiment was split into three separate tests: one concerning LLDs with SVMs, one concerning LLDs using CNNs, and one concerning mel-spectrograms with CNNs. The classifications using traditional SVMs were used as baseline to compare the results of the newer and less tested CNN procedure.

8.1. **Data Pre-Processing.** All audio recordings were loaded into a computer using the scripting language Python. All audio signals were sliced to remove silence at beginning and end of the recording. The data set thus at first consisted of raw audio vectors that each were split into a matrix of shape $A = [m, n]$, where m were the audio vectors and n were the 8 emotional classes. Furthermore, the matrix was organized to account for the 24 different speakers that each contributed equal amounts of audio recordings to the data set.
Every raw audio vector was first processed using STFT and the resulting vectors were subsequently used to extract audio features and finally subfeatures according to the algorithms outlined in section further above.

This, in turn, meant that standardization and normalization of every audio feature was done separately for each speaker, in accordance with standard procedure for speech emotion recognition [12]. After doing so, the data matrix was again transformed to combine all speakers per

emotional class, such that:

speaker$_i$	audio feature$_i$	audio feature$_{i+1}$	audio feature$_n$	emotion class$_i$
speaker$_{i+1}$	audio feature$_i$	audio feature$_{i+1}$	audio feature$_n$	emotion class$_i$
speaker$_n$	audio feature$_i$	audio feature$_{i+1}$	audio feature$_n$	emotion class$_i$

for all 94 LLDs extracted from the input audio signals and for all 8 classes. The resulting data matrix was shuffled randomly and split into training set and test set with an 80/20% partition.

8.2. **Low-Level Descriptors with Support Vector Machines.** A classification scheme was created where all input vectors were loaded from a matrix and fed into an SVM classifier. Model feature selection was calculated according to the formula [24]:

$$Var[X] = p(1-p)$$

where p is a probability value. If the p value is not significantly different from the mean, the feature is discarded. This means that the variance is maximized and only features that contribute new information to the model are used for classification. After perform model feature selection, 37 feature vectors of the original 94 were used per sample.

8.3. **Low-Level Descriptors with Convolutional Neural Networks.** The CNN used for further extracting features and classifying these was base on the VGG16 architecture [32]. A total of 25 layers were used, whereof the final five consisted of a fully connected vanilla neural network. The architecture can be visualized as:

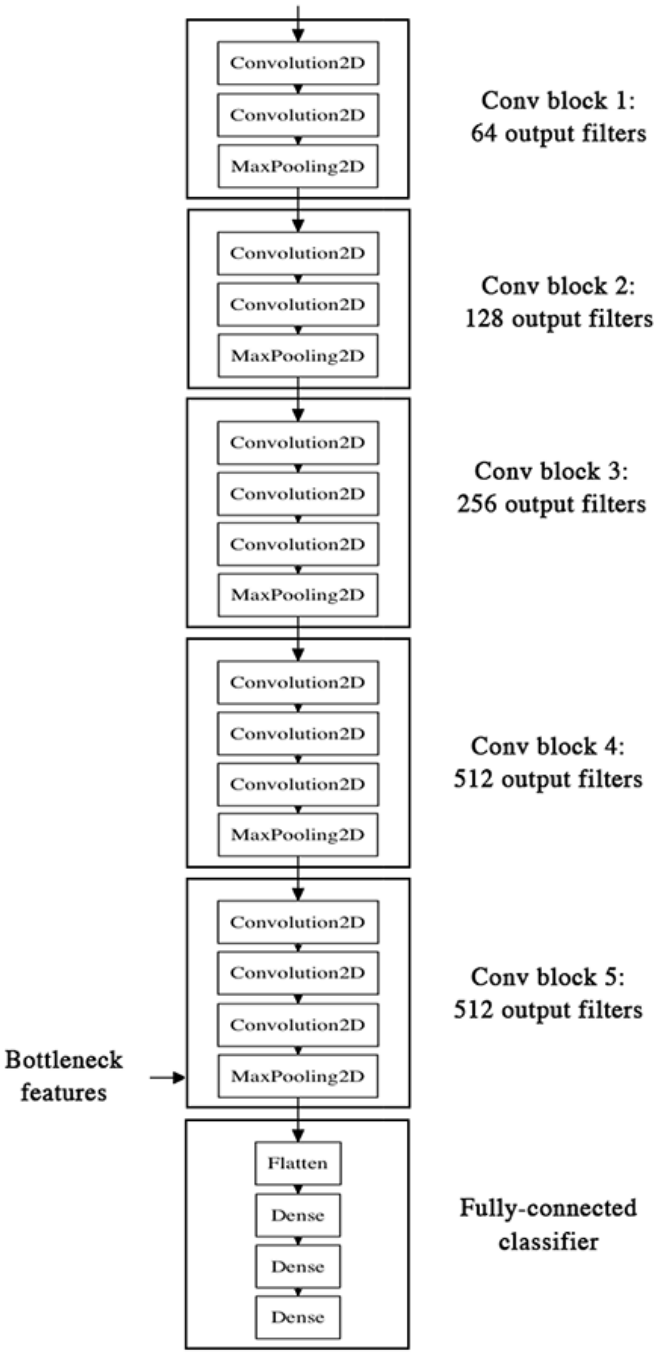


Figure from [33]: VGG16 CNN architecture.
Following feature extraction using the initial 20 layers of convolution, pooling, and batch normalization, the features were saved in a vector that subsequently was loaded into the fully connected top model for classification. With input dimensions ($n_{samples}$, 94, 1, 1) the entire data set could fit in memory, and there was no need to use generate mini batches. The model was trained and evaluated over 200 epochs.

8.4. **Mel-Spectrograms with Convolutional Neural Networks.** After calculating mel-spectrograms for every audio vector, as well as augmenting the data set with VTLP spectrograms, all the spectrograms were plotted and the resulting image matrices used for classification.

The resulting vectors contained pixel values of the dimensions 432 x 288 x 3, which were fed into the CNN. Multiple network architectures were tested using different optimization algorithms and the learning rate values 0.001, 0.01, and 0.1. With input dimensions 432 x 288 x 3, a data generator was built to fit the model, and mini batches of size 32 were feed to the network at a time.

8.5. **Model Evaluation.**

8.5.1. *F1 Score.* The performance of the multiclass SVM classifier was measured using average F1 score over all classes, with F1 calculated as:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

where precision is true positives over true positives and false positives:

$$\frac{tp}{tp + fp}$$

and recall is true positives over true positives and false negatives:

$$\frac{tp}{tp + fn}$$

8.5.2. *Categorical Cross-Entropy.* The performance of the CNN and its classifying fully connected network using a softmax function was measured using categorical cross-entropy. The cross-entropy is calculated by the loss function of the network, as explained in the methods section. It is important to note, that the cross-entropy accuracy is not updated with the rest of the model parameters during training. The cross-entropy accuracy is given as a mean accuracy of all classes for all predictions.

9. Results

9.1. **Low-Level Descriptors with Support Vector Machines.**

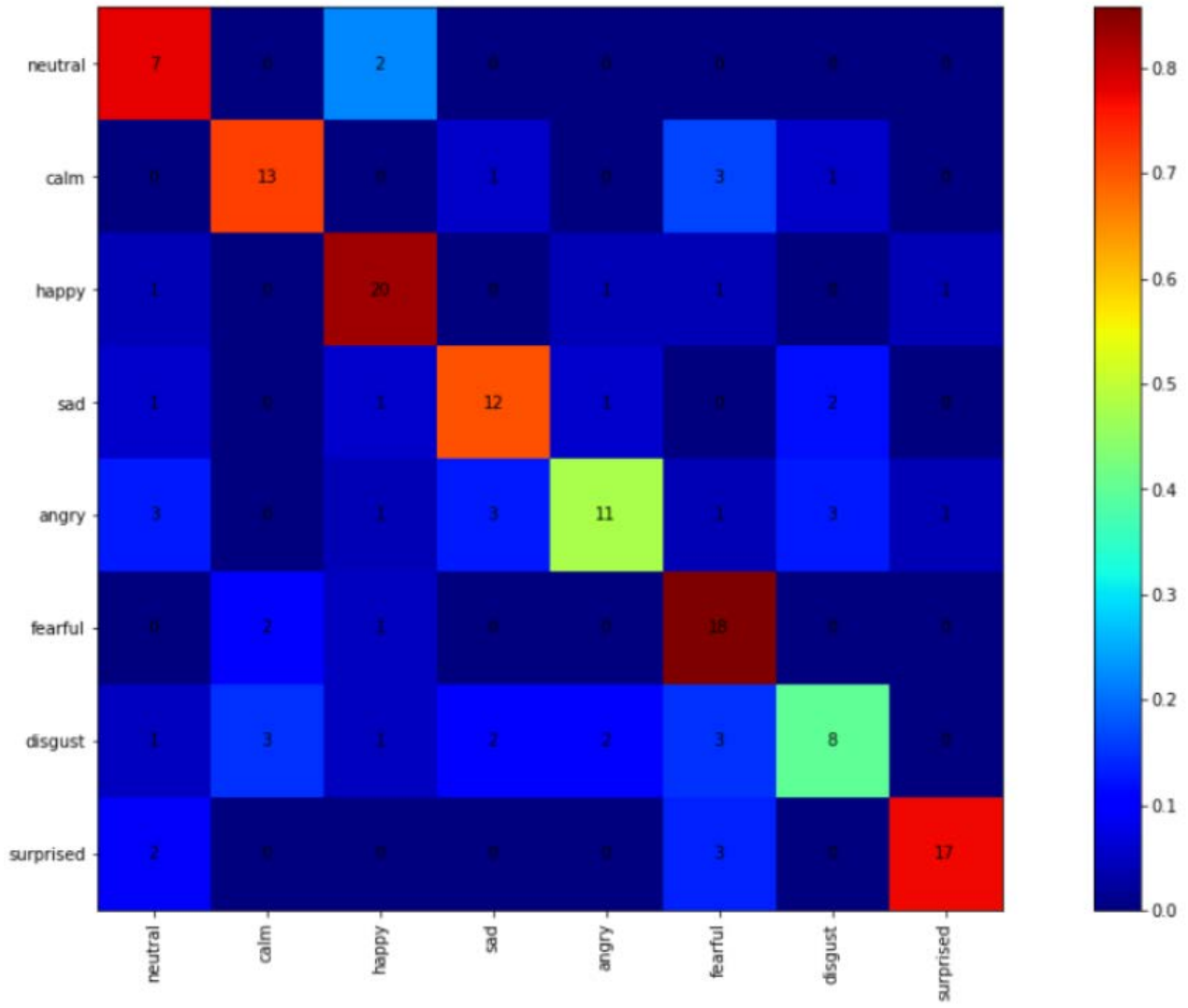
9.1.1. *RAVDESS.* The feature set containing LLDs from the RAVDESS data set was tested using an SVM classifier. The classification achieved an average accuracy of 0.68 across all classes. The average precision across all classes was 0.70 and the average recall across all classes was 0.69. The standard deviation from the predicted accuracies over all eight classes was 0.11. Student's t tests were calculated and the p value for each class compares the statistical difference from the average of all classes, where values < 0.05 are considered statistically significant. The precision, recall, and F1 score for all classes are listed in the table below:

Class	Precision	Recall	Accuracy	Accuracy p
Neutral	0.47	0.78	0.58	0.33
Calm	0.72	0.72	0.72	0.31
Happy	0.77	0.83	0.80	0.10
Sad	0.67	0.71	0.69	0.43
Angry	0.73	0.48	0.58	0.26
Fearful	0.62	0.86	0.72	0.27
Disgust	0.57	0.40	0.47	0.04
Surprised	0.89	0.77	0.83	0.07
Average	0.70	0.69	0.68	-
Standard Deviation	0.12	0.16	0.11	-

The class that achieved the largest precision value was *surprised* at 0.89, whereas the lowest precision was achieved for the *neutral* class at 0.47. The precision of the *surprised* class was the only one that was larger than the average precision of 0.70 with a value larger than the overall standard deviation of 0.12. The precision of the classes *neutral* and *disgust* were smaller than the average precision value with values larger than the standard deviation 0.12.

The class with the largest value for recall was *fearful*, whereas the lowest recall was found for the class *disgust*. The recall of the *fearful* class was larger than the average recall of 0.69 with a value larger than the standard deviation of 0.16. The recall of the classes *disgust* and *angry* were smaller than the average recall value with values larger than the standard deviation of 0.16.

The class with the best overall accuracy was *surprised*, whereas the lowest accuracy was calculated for *disgust*. The *disgust* class differs significantly from the average accuracy with $p = 0.04$, whereas *surprised* with $p = 0.07$ does not with $\alpha = 0.05$. However, with $\alpha = 0.1$, *surprised* would also be considered statistically significant. The class *happy* matches $\alpha = 0.1$. No other classes achieved accuracy levels that differed significantly from the mean accuracy.



Confusion matrix for the SVM classification of LLDs for RAVDESS.

The confusion matrix heat map visualizing predictions for the test set displays the most spread across classes for the *disgust* class, reflected by the lowest recall value. The classifications for the class *neutral* shows the greatest disparity between precision (0.47) and recall (0.78).

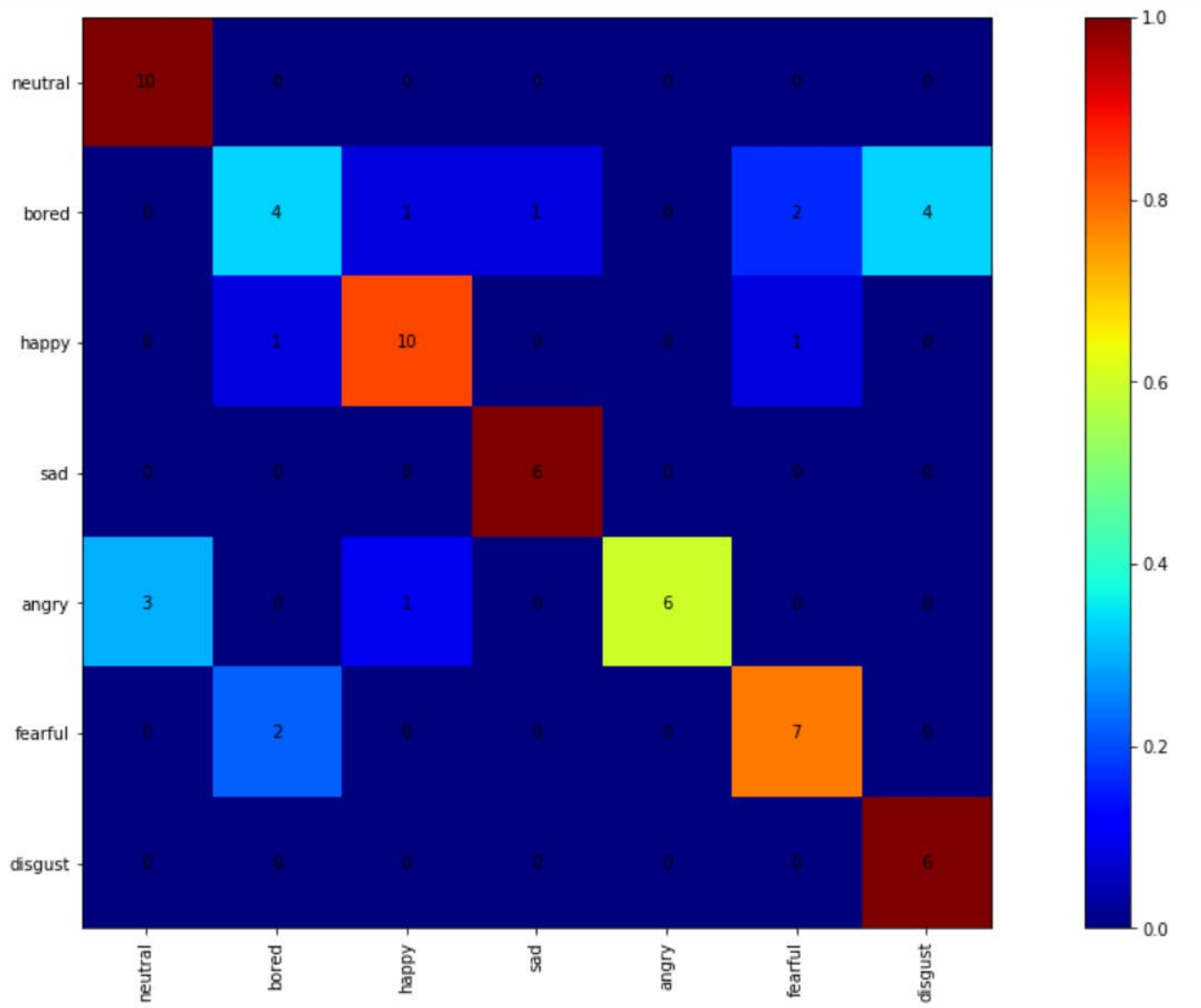
9.1.2. *Berlin emoDB.* The feature set containing LLDs from the Berlin emoDB data set was tested using an SVM classifier. Note that only seven classes were available Berlin emoDB set. The classes surprised and calm were not available for the Berlin emoDB data set, and the class bored was instead available. The amount of available recordings in the data set was unbalanced and hence only 46 audio files per class were randomly selected in order to balance the classes. The classification achieved an average accuracy of 0.74 across all classes. The average precision across all classes was 0.76 and the average recall across all classes was 0.75. The standard deviation from the predicted accuracies over all eight classes was 0.15. Student's t tests were calculated and the p value for each class compares the statistical difference from the average of all classes, where values < 0.05 are considered statistically significant. The precision, recall, and F1 score for all classes are listed in the table below:

Class	Precision	Recall	Accuracy	Accuracy p
Neutral	0.77	1.00	0.87	0.17
Bored	0.57	0.33	0.42	0.03
Happy	0.83	0.83	0.83	0.28
Sad	0.86	1.00	0.92	0.10
Angry	1.00	0.60	0.75	0.38
Fearful	0.70	0.78	0.74	0.45
Disgust	0.60	1.00	0.75	0.38
Average	0.76	0.75	0.74	-
Standard Deviation	0.14	0.23	0.15	-

The class that achieved the largest precision value was *angry* at a perfect 1.0, whereas the lowest precision was achieved for the *calm* class at 0.57. The precision of the *angry* class was the only one that was larger than the average precision of 0.76 with a value larger than the overall standard deviation of 0.14. The precision of the classes *calm* and *disgust* were smaller than the average precision value with values larger than the standard deviation 0.14.

The classes with the largest value for recall was *neutral*, *sad*, and *disgust* that all achieved a perfect 1.00; whereas the lowest recall was found for the class *calm*. The recall of the classes *neutral*, *sad*, and *disgust* were all larger than the average recall of 0.75 with a value larger than the standard deviation of 0.23. The recall of the class *calm* was smaller than the average recall value with a value larger than the standard deviation of 0.23.

The class with the best overall accuracy was *sad*, whereas the lowest accuracy was calculated for *calm*. The *calm* class differs significantly from the average accuracy with $p = 0.03$, whereas *sad* with $p = 0.10$ matches $\alpha = 0.1$. No other classes achieved accuracy levels that differed significantly from the mean accuracy.



Confusion matrix for the SVM classification of LLDs for Berlin emoDB.

The confusion matrix heat map visualizing predictions for the test set displays the most spread across classes for the *bored* class, reflected by the lowest recall value. The classifications for the class *angry* shows the greatest disparity between precision (1.00) and recall (0.60).

9.2. **Low-Level Descriptors with Convolutional Neural Networks.** The feature set containing LLDs (see section [CROSS-REFERENCE]) from the RAVDESS data set (see section [CROSS-REFERENCE]) was tested using an CNN classifier (see section [CROSS-REFERENCE]). The classification achieved an average accuracy of 0.79 across all classes. The classification algorithm of the CNN was evaluated with an average cross-entropy loss function across all classes and did not output individual accuracy levels for each class.

10. Discussion

10.1. **Low-Level Descriptors Comparison.** The SVM classifier for the Berlin emoDB set achieved an accuracy of 74%, yet the data set was too small to use for the CNN. The SVM

classifier for the subfeature set from the RAVDESS data set achieved an average accuracy of 68%, and the convolutional neural network achieved an average accuracy of 79%.

The best accuracy was thus achieved by the CNN for the RAVDESS set. While the mAP differed with 11 percentage points in favour of the CNN—a considerable value when evaluating classification algorithms—a student's t-test was calculated and with a p value = $0.16 > \alpha = 0.05$, there was no statistically significant difference between the two models on the data set. The CNN for RAVDESS set was 5 percent points better than the SVM for Berlin emoDB, but with a p value = $0.36 > \alpha = 0.05$, there was no statistically significant difference. The two SVMs on both data sets differed by 6 percentage points in favour of the Berlin emoDB set, but with a p value = $0.29 > \alpha = 0.05$, there was no statistically significant difference between the classifiers.

10.2. **Low-Level Descriptors Compared to Previous Studies on RAVDESS.**

10.2.1. *Comparison to Human Raters.* In the paper accompanying the RAVDESS data set [REFERENCE], human raters achieved an average accuracy of 62%, which was beaten by both the SVM and the CNN. The SVM classifier did not achieve a statistically significant difference with a p value = $0.16 > \alpha = 0.05$, but the CNN did with p value = $0.04 < \alpha = 0.05$.

10.2.2. *Comparison to Previous SVM Study.* In the paper by Zhang, et al [12], testing SVMs on LLDs from the RAVDESS set, an average accuracy of 80% was achieved. It should however be noted, that only six classes: angry, happy, neutral, sad, calm, and fearful were used. In the present study, the SVM classification of the RAVDESS LLDs achieved only 68%, a difference of 12 percentage points. However, with a p value of 0.05, it matches an $\alpha = 0.05$, but cannot be said to be statistically significant. The CNN used in the present study achieved an accuracy of 79%, a difference in only 1 percentage point, which gives a p value of $0.35 > \alpha = 0.05$.

10.3. **Mel-Spectrograms with Convolutional Neural Networks.** The mel-spectrograms with dimensions 432 x 288 x 3 proved to be too computationally expensive for an i7 CPU with 16 GB RAM to properly handle. While transfer learning using weights from the ImageNet [32] and other models from [33] was implemented and tested, the much smaller size of the RAVDESS data set lead to overfitting, where training set achieved accuracy levels near 1.0 within 10 epochs. The evaluation and test sets, however, never reached accuracy levels above 0.33. To account for this, learning rates were lowered to 0.001 but without the proper computational power of a GPU, model training of over 2 weeks were not possible to handle without running out of memory.
Without transfer learning from a pre-trained network, the model was not able to properly read enough data from the image vectors and achieved average accuracy levels at 0.15, here interpreted as chance over eight classes.
Smaller models of no more five layers were equally tested, but with over 5 million model parameters for input dimensions 432 x 288 x 3 for only 3072 input vectors (original and augmented image vectors), even the smaller models lead to overfitting, regardless of optimization, learning rate, or dropout values.

11. Conclusions

While the CNN feature extractor and classifier for the sparse RAVDESS LLD subfeature set achieved the best accuracy of all tested classifiers, no statistically significant difference could be found against the SVM classifier in the study. It did however prove to be a better classifier than human raters.
Much effort was put into the mel-spectrogram feature set extracted and augmented from the RAVDESS data set, but the lack of computational power in the form of a GPU rendered the experiment unsuccessful. Furthermore, a much larger data set would be needed to fully make use of the possibilities of CNNs, yet a lack of publicly available data for emotional speech makes this problematic. A suggested solution would be further data augmentation, such as e.g. random slicing of the audio files. However, in the data set used, the average duration of the recordings was ca 3 seconds before slicing silence at end and beginning. Further slicing could lead to a lack of useful data if e.g. pitch envelope across the recording could not be extracted.

12. References

[1] Hinton, G.E., et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition. In *Signal Processing Magazine, IEEE*, vol. 29, no. 6: 82–97. 2012.

[2] Schlüter, J. & Böck, S. Improved Musical Onset Detection with Convolutional Neural Networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6979-6983, 2014.

[3] Wu, Z., Valentini-Botinhao, C., Watts, O., & King, S. Deep neural networks employing Multi-Task Learning and stacked bottleneck features for speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4460-4464, 2015.

[4] Richardson, F., Reynolds, D.A., & Dehak, N. Deep Neural Network Approaches to Speaker and Language Recognition. In *IEEE Signal Processing Letters 22*, pp. 1671-1675, 2015.

[5] Yoon, S., Byun, S., & Jung, K. Multimodal Speech Emotion Recognition Using Audio and Text. arXiv preprint arXiv:1810.04635. 2018.

[6] Mattheson, J. Harriss, E.C., ed., *Johann Mattheson's Der vollkommene Capellmeister: A Revised Translation with Critical Commentary. Studies in Musicology, no. 21.* Ann Arbor: UMI Research Press, 1981.

[7] Sendlmeier, W.F. & Klasmeyer, G. Voice and Emotional States. In *Voice Quality Measurement*, pp. 339-357. Singular, San Diego, CA. 2000.

[8] Paeschke, A. & Sendlmeier, W.F. Prosodic Characteristics of Emotional Speech: Measurements of Fundamental Frequency Movements". In *Proceedings of the ISCA Workshop on Speech and Emotion*, pp. 75-80. Textflow, Belfast, Northern Ireland, 2000.

[9] Kienast, M. & Sendlmeier, W.F. Acoustical Analysis Of Spectral And Temporal Changes In Emotional Speech. In R. Cowie, E. Douglas-Cowie, M. Schroeder, eds. *Speech and Emotion: Proceedings of the ISCA workshop. Newcastle, County Down, Sept. 2000*, pp. 92-97, Belfast,

Northern Ireland, 2000.

[10] Kienast, M., Paeschke, A., & Sendlmeier, W.F. Articulatory reduction in emotional speech. In *Proceedings Eurospeech 99, Budapest, Vol. 1*, pp. 117-120, 1999.

[11] Schuller, B., Steidl, S., Batliner, A., & Jurcicek, F. The INTERSPEECH 2009 emotion challenge. In *Proceedings 10th Annual Conference International Speech Communication Association*, pp. 312-315, Sep. 2009.

[12] Zhang, B., Essl, G., & Mower Provost, E. Recognizing emotion from singing and speaking using shared models. In *Proceedings of Affective Computing and Intelligent Interaction*, 2015.

[13] Triantafyllopoulos, A. Sagha, H., Eyben, F., & Schuller, B. audEERING's approach to the One-Minute-Gradual Emotion Challenge. arXiv preprint arXiv:1805.01222. 2019

[14] Mao, Q., Dong, M., Huang, Z. & Zhan, Y. Learning Salient Features for Speech Emotion Recognition Using Convolutional Neural Networks. In *IEEE Trans. Multimedia 16*, 8, pp. 2203–2213, 2014.

[15] Cummins, N., Amiriparian, S., Hagerer, G., Batliner, A., Steidl, S. & Schuller, B. An Image-based Deep Spectrum Feature Representation for the Recognition of Emotional Speech. In *Proceedings of the 25th ACM International Conference on Multimedia MM 2017, October 23–27, 2017, Mountain View, CA, USA*, pp. 7, 2017.

[16] Lerch, A. *An Introduction to Audio Content Analysis: Applications in Signal Processing and Music Informatics*. Hoboken, New Jersey: John Wiley & Sons, 2012.

[17] McFee, B., McVicar, M., Raffel, C., Liang, D., Nieto, O., Moore, J., Ellis, D., et al. *Librosa: v0.4.0*. Zenodo, 2015. doi:10.5281/zenodo.18369

[18] Stevens, S.S.; Volkmann; J.. & Newman, E.B. A scale for the measurement of the psychological magnitude pitch. In *Journal of the Acoustical Society of America*. 8 (3): 185–190. 1937.

[19] Jurafsky, D., & Martin J.H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd Ed. Upper Saddle River, New Jersey: Pearson, 2009.

[20] Jiang, D.N., Lu, L., Zhang, H.J., Tao, J.H, Cai, L.H. Music Type Classification by Spectral Contrast Feature. In *International Conference on Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE*, Volume: 1. 2002.

[21] Böck, S., & Widmer, G. *Maximum filter vibrato suppression for onset detection.* 16th International Conference on Digital Audio Effects, Maynooth, Ireland. 2013.

[22] Grosche, P., Müller, M. & Kurth, F. *Cyclic tempogram - A mid-level tempo representation for music signals.*, ICASSP, 2010.

[23] James, G., Witten, D., Hastie, T., & Tibshirani, R. *An Introduction to Statistical Learning*. New York: Springer, 2013.

[24] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. Scikit-learn: Machine Learning in Python. In *Journal of Machine Learning Research*, 12, pp. 2825-2830, 2011.

[25] Burkov, A. The Hundred Page Machine Learning Book. Self print. 2019.

[26] Charniak, E. Introduction to Deep Learning. MIT Press, Cambridge, Massachusetts. 2018.

[27] Goodfellow, I., Bengio, Y., & Courville, A. *Deep Learning*. Cambridge, MA: MIT Press, 2016. Available online: https://www.deeplearningbook.org/

[28] Ioffe, S., & Szeedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv:1502.03167. 2015.

[29] Jaitley, N., Hinton, G.E. *Vocal Tract Length Perturbation (VTLP) improves speech recognition*. In Proceedings of the 30th International Conference on Ma-chine Learning, Atlanta, Georgia, USA, 2013.

[30] Livingstone, S.R. & Russo, F.A. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. *PLoS ONE 13(5): e0196391*, 2018.

[31] Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W.F., & Weiss, B. A Database of German Emotional Speech. In *Proceedings Interspeech, Lissabon, Portugal*. 2005.

[32] Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR abs/1409.1556. 2014.

[33] Chollet, Francois, et al., 2015 Keras. https://keras.io