Voice Feature Extraction Using Siamese Neural Networks for Detecting Impersonators

Marlon Almström, Hoa Tran

Master's thesis
2022:E35

Lund University

Faculty of Science
Centre for Mathematical Sciences
Mathematical Statistics

Abstract

Voice impersonation is a technique that has often been used by criminals whose goal is to avoid being identified while committing a crime. There are, however, other interesting cases where the police confronts a suspect with an incriminating recording, and the suspect would deny being the true speaker in that recording, and claim that it belonged to an expert impersonator. In both of these cases, it would be very helpful for the police to be able to predict with high probability whether a recording belongs to the true speaker or an impersonator.

This thesis aims to use neural networks to extract the most significant features in recognizing a unique voice, and then use them to classify whether a recording belongs to a true speaker or somebody impersonating them. In order to achieve this, we first extract the raw audio features that are commonly used in speech recognition, the majority of which are spectral features, then feed these features to a Siamese Neural Network to generate an encoding that best represent a recording of a person's voice. The structure of a Siamese neural network is determined by the type of loss function being used. In this project, we compare the performances of different network structures as well as different classifiers used in classifying the speech from the encoding.

We present our approach and results on the data consisting of recordings of prominent American political figures, their impersonators, and several other individuals.

Acknowledgments

We would like to express our deep gratitude to our supervisor, Professor Andreas Jakobsson, at the Centre for Mathematical Sciences, Mathematical Statistics at Lund University. The project has been challenging but also very fun. We have really enjoyed it and learned a lot. All of that is thanks to your guidance, support and interesting discussions. Thank you also for the funny jokes that cracked us up even when the thesis was not going well.

We would also like to thank Susanna Whitling for sharing her expert knowledge, which directed us to experiment new features that we could not have come up with ourselves.

Glossary

ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
AUC	Area Under the Curve
DFT	Discrete Fourier Transform
DTFT	Discrete-Time Fourier Transform
F0	Fundamental Frequency
HNR	Harmonic to Noise Ratio
LPC	Linear Predictive Coding
MFCCs	Mel Frequency Cepstral Coefficients
PSD	Power Spectral Density
ROC	Receiver Operating Characteristic
SNN	Siamese Neural Network
STFT	Short-Time Fourier Transform
t-SNE	t-distributed Stochastic Neighbor Embedding

Contents

1 Introduction

1.1 Background

People are most often able to tell who is present even before seeing them if their voice is heard. Our voices are one of the most common things that help others identify us. However, if a person does not want to be identified by their voice, they can try to mimic a different accent or imitate another person. Therefore, voice impersonation is a subject in voice analysis in criminal forensics.

Stemming from a real-life problem: voice impersonation has been used for the purpose of faking the identity of another person when a crime is committed. Moreover, it could also be the case that the police gets hold of an incriminating recording which they think belongs to a suspect; however, the speaker denies owning it and claims that it belongs to an expert impersonator [21]. In both cases, it would be very helpful to be able to verify whether a recording belongs to its true speaker or an expert impersonator of that speaker.

In order to answer this question, it is useful to understand what constitutes a person's unique voice. As expected, many factors are involved in the production of a person's unique voice. It starts in the lungs, where air is exhaled to create an airstream in the trachea and across the larynx, i.e., the sound box. The vocal folds, also known as vocal cords are stretched horizontally across the larynx. As air passes over them, the vocal cords vibrate very quickly to produce sounds. This process is illustrated in Figure 1. It is the vibration of the vocal cord that produces the periodic structures in our voiced signals. However, since the vocal cord is an organic structure, the signals are not perfectly periodic, but contain also fluctuations. Based on these almost periodic structures, the fundamental frequency of a person's voice can be computed as the average number of oscillations per second. The fundamental frequency, often denoted as F0, is closely related to the pitch, which is defined as our perception of fundamental frequency [3]. This means that the F0 describes the actual physical phenomenon based on the signal, whereas pitch describes how our ears and brains interpret the signal, in terms of periodicity. While the fundamental frequency of one's voice is mainly influenced by the length and tension of the vocal cords, the other parts such as the throat, nose, and mouth act as a resonating chamber for the buzzing sounds made by the vocal cords, turning them into a unique voice. Therefore, in order to study what constitutes the uniqueness of a person's voice, it is natural to look into the periodic structures of the voiced signals and extract features from there. As observed in our project, the fundamental frequency turns out to be an integral part when computing different features of a voiced signal.

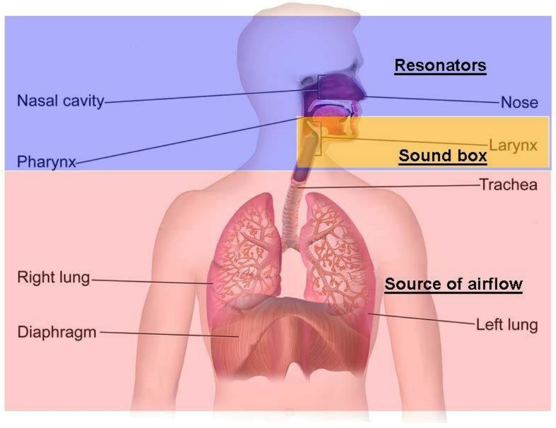


Figure 1: Voice production system: thoracic cavity (airflow source), larynx (sound box) and vocal tract (resonator). Taken from “Blausen gallery 2014". Wikiversity Journal of Medicine.

Many of the features that were tested in our project are common features in speech recognition such as spectrograms, Mel Frequency Cepstral Coefficients (MFCCs), formants, formant ratios, bandwidths [26].
Machine learning is becoming more popular in many different fields for the tasks of identification and classification. However, it often requires access to a large amount of data in order to achieve meaningful results. In real life, for many different tasks, the availability of the data is often the limiting factor. Moreover, even when one has access to enough data, processing a large amount of data also poses a problem of high computational cost. To tackle these problems, many researchers have worked on creating machine learning models that do not require as much data but can still deliver similar results.
An approach to structure an artificial neural network (ANN) that has shown to perform well on a smaller amount of data is the Siamese Neural Network (SNN) [16], introduced for the task of one-shot image recognition. One-shot learning is a classification task where the model only uses one or a few samples in training in order to predict the class of the test image. The name Siamese Neural Network stems from the fact that SNN consists of two or more identical neural networks, which are trained simultaneously. What the SNN does is to produce vector encodings of its inputs where the goal is to separate the encodings of different classes (which may or may not have some similarities), while simultaneously trying to cluster the encodings of same classes. In order to determine which classes these encodings belong to, it is common to use some similarity metric, such as the cosine similarity. In this project, the k-Nearest Neighbor (kNN) and random forest classifiers are used for classifying the encodings. Examples of inputs where the SNN has shown to perform well are image classification, as well as audio classification [27]. For our thesis, voice classification is the focus.

1.2 Problem formulation

We aim to answer the following questions:

- What audio features are useful in detecting a person's unique voice?
- What network architectures are most effective in detecting an impersonator's voice?

The data used for the project are voices of prominent American politicians, their impersonators, as well as some other random people. Using this data set, we investigate a number of different combinations of features from spectral analysis together with different network architectures as well as different classification algorithms.
The structure of the thesis starts with the theory, in which, the basic theory on what the audio features really are, is presented first, together with some important aspects that clarifies how some of the features are extracted. After that, more details about the machine learning part, i.e., what is needed in order to build an artificial neural network, as well as descriptions of the different structures of the Siamese Neural Network, are presented. Once all the theory is presented, the methods used in order to achieve our results, such as the data collection, data processing, model performance evaluation are explained. Thereafter, the results are presented to show how well the networks and classifiers manage to classify the true speaker, their impersonator, and random people in two separate cases, one with Donald Trump as the true speaker and the other one with Barack Obama.

2 Theory

2.1 Audio features

2.1.1 Spectrogram

The periodogram is an estimate of the Power Spectral Density (PSD), which is defined as the discrete-time Fourier transform (DTFT) of the auto-covariance function [14], i.e., for $-\pi < \omega \leq \pi$,

$$\phi_y(\omega) := \sum_{k=-\infty}^{\infty} r_y(k)e^{-i\omega k}. \tag{1}$$

Under the assumption that $\lim_{N\to\infty} \frac{1}{N}\sum_{k=-N}^{N} |k||r_y(k)| = 0$, the PSD can be expressed as

$$\phi_y(\omega) = \lim_{N\to\infty} E\left(\frac{1}{N}\left|\sum_{t=1}^{N} y_t e^{-i\omega t}\right|^2\right). \tag{2}$$

Using the discrete Fourier transform (DFT), i.e.,

$$Y(k) = \sum_{t=1}^{N} y_t e^{-i2\pi kt/N}, \tag{3}$$

suggests a natural estimator of the PSD known as the periodogram which is obtained by the magnitude square of the DFT, i.e.,

$$\hat{\phi}_y(k) = \frac{1}{N}|Y(k)|^2 = \frac{1}{N}\left|\sum_{t=1}^{N} y_t e^{-i2\pi kt/N}\right|^2. \tag{4}$$

It is well known that the periodogram has large variance and bias, and is therefore not always the best choice when the goal is to obtain a spectral estimate. The idea of filtering the periodogram through several different windows was introduced by Thomson [29], and has been shown to outperform other methods such as periodogram and Welch method in terms of leakage, variance, and resolution. The multiple window spectrum estimate is defined as

$$\hat{\phi}_y(k) = \frac{1}{K}\sum_{n=1}^{K}\left|\sum_{t=1}^{N} y_t h_n(t) e^{-i2\pi kt/N}\right|^2, \tag{5}$$

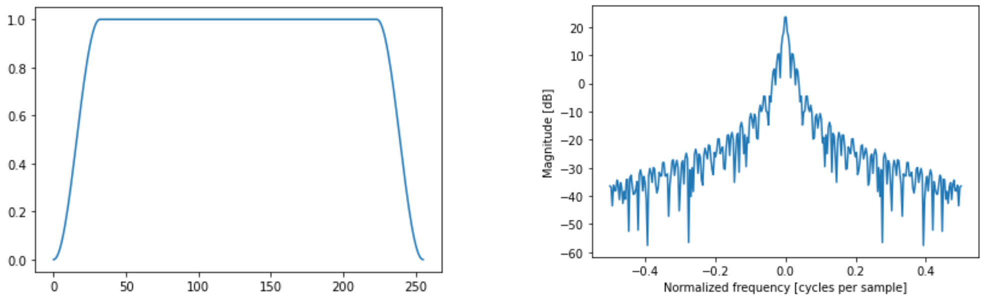
where h_n are the specific windows one decides to use. The windows that were chosen in this thesis were the Tukey (tapered cosine) windows [4] and sinusoidal windows [12]. The Tukey windows are defined as

$$h_n(t) = \begin{cases} \frac{1}{2}\left(1+\cos(\frac{2\pi}{r}[t-r/2])\right) & , \ 0 \leq t < \frac{r}{2} \\ 1 & , \ \frac{r}{2} \leq t < 1-\frac{r}{2} \\ \frac{1}{2}\left(1+\cos(\frac{2\pi}{r}[t-1+r/2])\right) & , \ 1-\frac{r}{2} \leq t < 1 \end{cases},$$

where the shape parameter r is chosen as $r = 0.25$. Unlike the Tukey windows, the sinusoidal windows use different windows (multitapering) and are defined as

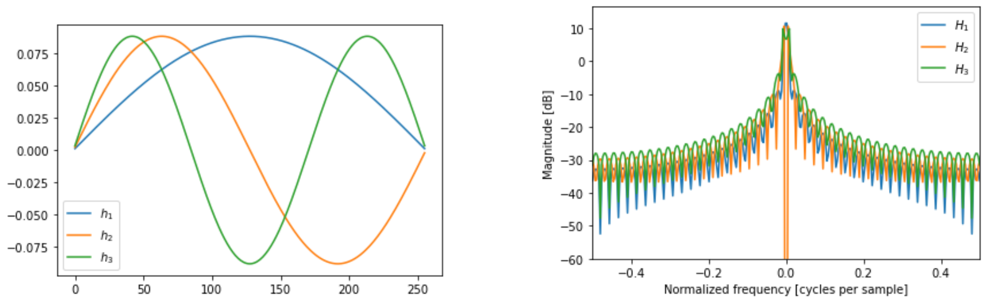
$$h_n(t) = \sqrt{2/N sin\left(\frac{\pi n \cdot t}{N}\right)}.$$

The Tukey window together with its frequency response can be seen in Figure 2, and an example of how three sinusoidal windows can look like together with their corresponding frequency responses can be seen in Figure 3.



(a) Tukey window **(b)** Window spectrum

Figure 2: Tukey window and its frequency response.



(a) Sinusoidal windows **(b)** Window spectrum

Figure 3: Sinusoidal windows and their corresponding frequency responses.

When signals are time-varying or non-stationary, the spectrogram is a commonly used tool. It is formed using the short-time Fourier transform (STFT), which is defined as

$$Y(t, \omega) = \int_{-\infty}^{\infty} y_{t_1} h^*(t_1 - t) e^{-i\omega t_1} dt_1, \tag{6}$$

where $h(t)$ is a window function centered at time t [24]. The window cuts the signal around a small neighbourhood around t, and then the Fourier transform is estimated around this neighbourhood. The spectrogram is then obtained by taking the absolute value squared of the STFT, i.e.,

$$\phi(t, \omega) = |Y(t, k)|^2. \tag{7}$$

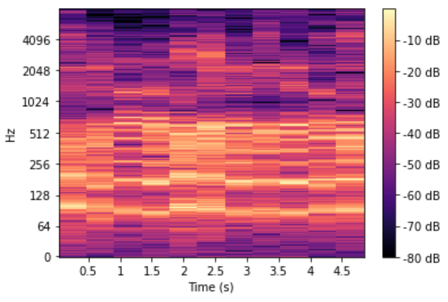
However, the signal is usually sampled with a sample distance T, i.e., $y_t = y(tT)$, where $T = 1/F_s$ (F_s being the sample frequency), which then gives the discrete-time and discrete-frequency spectrogram as

$$\phi(t,k) = \left| \sum_{t_1=1}^{N} y_{t_1} h^*(t_1 - t + M/2) e^{-i2\pi k t_1/N} \right|^2, \tag{8}$$

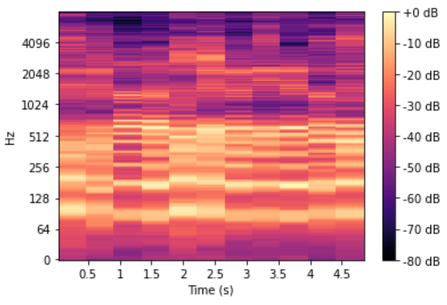
where M is the length of the window $h(\cdot)$. The same idea of multitapered spectrums can be applied to the spectrogram, resulting in the multitapered spectrogram, defined as:

$$\phi(t,k) = \frac{1}{K} \sum_{j=1}^{K} \left| \sum_{t_1=1}^{N} y_{t_1} h_j^*(t_1 - t + M/2) e^{-i2\pi k t_1/N} \right|^2. \tag{9}$$

An example of the spectrogram of a five-second recording split into 11 time segments can be seen in Figure 4, where in Figure 4a the regular spectrogram using a Tukey window is shown, and in Figure 4b the multitapered spectrogram using 8 sinusoid windows for each time segment is shown.



(a) Tukey window



(b) Sinusoid windows

Figure 4: Spectrogram using Tukey window in (a), and multitapered spectrogram using 8 sinusoid windows in (b).

Both the optimal window length and the optimal number of windows are hard to choose, since using longer windows will result in higher resolution in frequency but worse resolution in time, while using shorter windows will result in worse resolution in frequency but higher resolution in time. For this project, the difference in frequency may be what distinguishes one person from another, hence, the window length was chosen rather large to increase the frequency resolution.

2.1.2 Mel spectrogram

Mel scale

The Mel Scale is the result of a non-linear transformation of the frequency scale in Hz in order to detect frequency differences the same way as how a human perceives sounds. For example, the human ear can easily hear the difference between 500 and 1000 Hz, whereas the difference between 7500 and 8000 Hz is barely noticeable, even though the absolute distance between the frequencies is the same. The mel scale is used to mimic the non-linear human ear perception of sound and does so by being more discriminative at lower frequencies and less discriminative at higher frequencies.

The transformation going from frequency f to Mel scale f_{mel} is

$$f_{mel} = 2595\log_{10}\left(1 + \frac{f}{700}\right). \tag{10}$$

The Mel spectrum is obtained by filtering the magnitude square of the DFT through a filterbank which contains a pre-specified number of windows. The most commonly used filterbank consists of triangular shaped windows, as shown in Figure 5 and are defined as

$$H_m(k) = \begin{cases} 0 & , \; k < f(m-1) \\ \frac{k-f(m-1)}{f(m)-f(m-1)} & , \; f(m-1) \leq k \leq f(m) \\ \frac{f(m-1)-k}{f(m+1)-f(m)} & , \; f(m) \leq k \leq f(m+1) \\ 0 & , \; k > f(m+1) \end{cases},$$

for $m = 1, ..., M$ (M being the number of filters), where $f(\cdot)$ are the $M+2$ Mel-spaced frequencies.

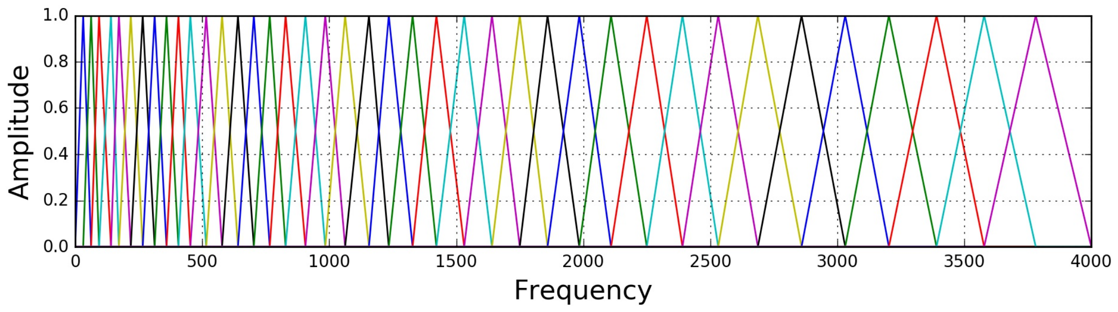


Figure 5: Filter bank on a Mel-Scale [23]

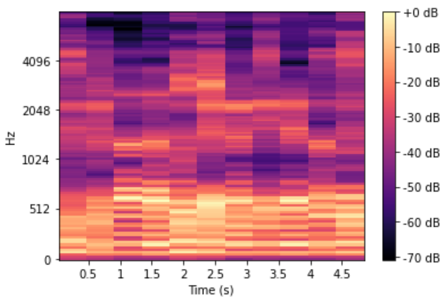
Now that the filterbank has been introduced, together with equation (3), the Mel spectrum can be defined as

$$\phi_{mel}(m) = \sum_{k=1}^{N} |Y(k)|^2 H_m(k), \quad 1 \leq m \leq M. \tag{11}$$

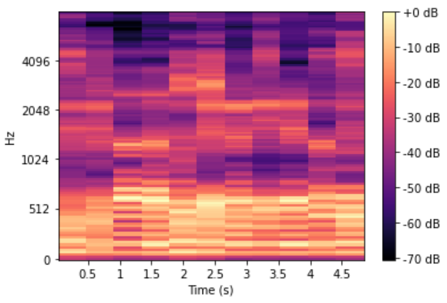
The same idea is used when computing the Mel spectrogram, i.e., using equation (8), the Mel spectrogram is defined as

$$\phi_{melS}(t,m) = \sum_{k=1}^{N} \phi(t,k) H_m(k), \quad 1 \leq m \leq M. \tag{12}$$

An example of how the Mel spectrogram on the same recording shown in Figure 4 can be seen in Figure 6, where the difference between the Tukey and the multitapered mel spectrogram is not as obvious as for the regular spectrogram.



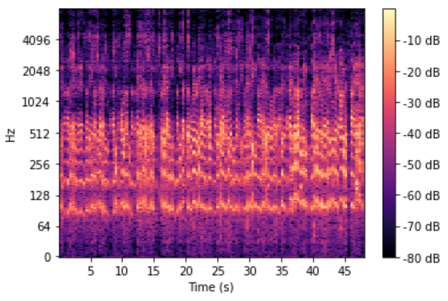
(a) Tukey window



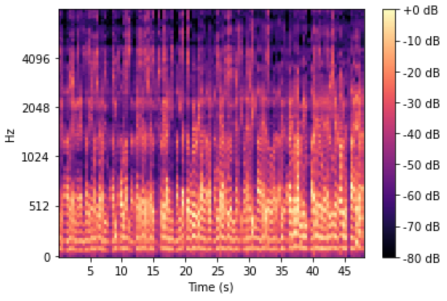
(b) Sinusoid windows

Figure 6: The mel spectrogram using a Tukey window in (a), and the multitapered mel spectrogram using 8 sinusoid windows in (b).

Another example comparing the spectrogram and mel spectrogram can be seen in Figure 7, where a longer recording (48s) have been chosen.



(a) Spectrogram



(b) Mel spectrogram

Figure 7: The spectrogram (a) and the Mel spectrogram (b) of a recording of someone speaking for 48 seconds.

The Mel frequency cepstral coefficients (MFCCs) have been shown to be good features for automatic speech recognition (ASR). They are obtained by computing the discrete cosine transform (DCT) of the log filterbank energies [23], i.e., using equation (11), the MFCCs are defined as

$$c(n) = \sum_{m=1}^{M} \log_{10}(\phi_{mel}(m)) \cos\left(\frac{\pi n(m-0.5)}{M}\right), \qquad n = 1, ..., C \tag{13}$$

where C are the number of MFCCs by design. Typically, for ASR, the first coefficient is discarded since it represents the average log-energy of the input signal, which does not carry very much speaker-specific information, and only the next twelve MFCCs are used. The other higher order terms are excluded as they represent fast changes in the filter bank coefficients which do not contribute to the performance of an ASR system.

2.1.3 Fundamental Frequency and Formants

To produce speech, a person tenses the vocal folds to make it oscillate in the airflow coming from the lungs. The fundamental frequency ($F0$) is defined as the average number of oscillations per second and is expressed in Hertz, often between 80 and 450 Hz, where males have lower fundamental frequencies than females and children [3].

The fundamental frequency is the peak with the smallest frequency in the spectrum which also has peaks located at the integer multiples of the fundamental frequency, as seen in Figure 8. For our project, $F0$ is computed using the `Signal_Analysis` package in Python. In this package, the fundamental frequency is estimated by segmenting a given signal into frames, then the median of the estimated $F0$'s from each frame is chosen as the final estimate of $F0$. The algorithm used is adapted from [5].

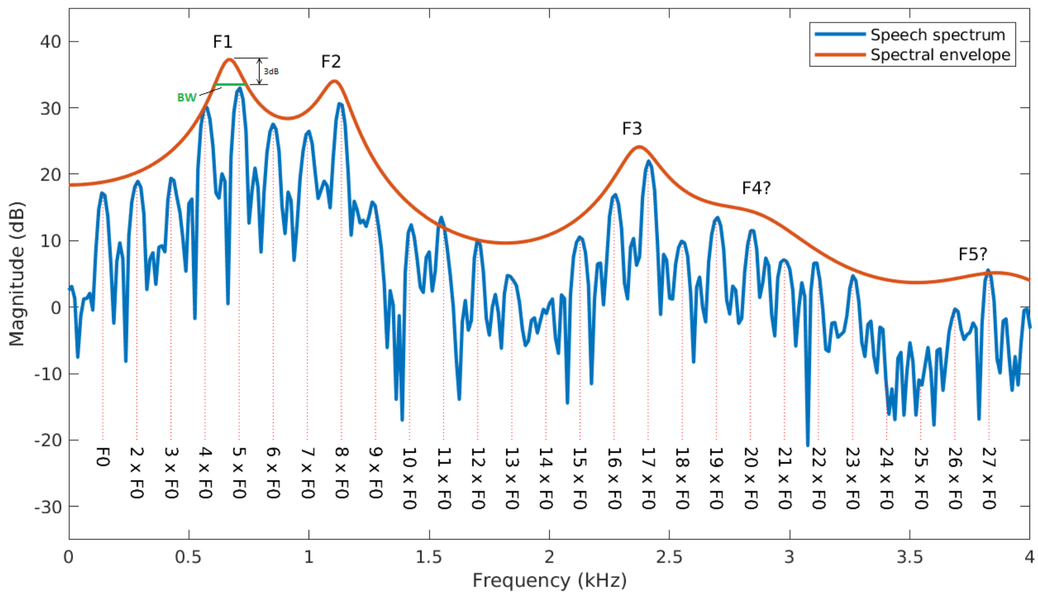


Figure 8: Fundamental frequency and formants seen from spectrum of a speech signal. The figure is reproduced from [3].

Formants, often denoted as $F1$, $F2$, $F3$, $F4$, etc., refer to the different frequency peaks in a speech spectrum, as seen in Figure 8. To create the spectral envelope for estimating the formant frequencies, one can use linear predictive coding (LPC) [1]. In Figure 8, under the first formant F1, a formant bandwidth is illustrated by the green line that is marked BW for bandwidth. Specifically, the formant bandwidth is measured, by convention, 3 dB below the peak.

2.1.4 Sinusoidal fitting

The phonetic definition of a vowel is a sound produced with no constriction in the vocal tract, resulting in the periodic structures often present in a vowel. The goal then is to capture the uniqueness of each person's voice by extracting features from the harmonic structures in the vowel parts of the audio. In order to capture the harmonic structures in the audio signals, sinusoidal fitting of the signal is used to determine if the signal is periodic enough. For fast and robust computation, the fundamental frequency of the signal is estimated, and used as a known frequency in the Fourier matrix (the A matrix below) for least square optimization. If the assumption is that the frequencies are not known, a d-dimensional search is required, which could give better fitting but also take much longer time to compute, compared to when the fundamental frequency is assumed to be known.
To further save computational cost, the assumption that the fundamental frequency has the highest amplitudes of all the peaks in the periodogram is made, which is usually the case, but is not necessarily so. The sine fitting is computed for the 20- and 40-millisecond segments of the audio files. Assume the model for a 20- or 40-millisecond signal y_t:

$$y_t = \sum_{k=1}^{d} \alpha_k e^{i\omega_k t + i\phi_k} + e_t, \quad t = 1, 2, ..., N,$$

which can be written as $\mathbf{y} = [y_1, ..., y_N]^T = Ax + e$, where

$$A = \begin{bmatrix} e^{i\omega_1} & e^{i\omega_2} & e^{i\omega_3} & \dots & e^{i\omega_d} \\ e^{i\omega_1 \cdot 2} & e^{i\omega_2 \cdot 2} & e^{i\omega_3 \cdot 2} & \dots & e^{i\omega_d \cdot 2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e^{i\omega_1 \cdot N} & e^{i\omega_2 \cdot N} & e^{i\omega_3 \cdot N} & \dots & e^{i\omega_d \cdot N} \end{bmatrix}, \quad \text{and} \quad x = \begin{bmatrix} \alpha_1 e^{i\phi_1} \\ \alpha_2 e^{i\phi_2} \\ \vdots \\ \alpha_d e^{i\phi_d} \end{bmatrix}.$$

Here, it is assumed that ω_1 is the estimated fundamental frequency F0 and ω_k, for $k = 2, 3, ..., d$ are the k multiples of ω_1, $d = 10$. The amplitudes can then be solved for in closed form as

$$\hat{x} = \underset{x}{\operatorname{argmin}} ||\mathbf{y} - Ax||_2^2.$$

After fitting the segment, the fitting is evaluated using the R^2 score [10], which is defined as the sum of squared fitted-value deviations divided by the sum of squared original-value deviations, i.e.,

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2},$$

where $\hat{y}_i$ is the sine fit at each time t, and $\bar{y}$ is the mean of the signal y_t. In the best case, the fitted values exactly match the observed values, which results in $R^2 = 1$ and in the worst case, where the predicted $\hat{y}$ is just the average $\bar{y}$, $R^2 = 0$.

2.1.5 Other features

The Harmonic to Noise Ratio (HNR) measures the ratio between periodic and non-periodic components of a speech sound. A speech signal $x(\omega)$ in the frequency domain can be modelled as:

$$x(\omega) = h(\omega) + n(\omega),$$

where $h(\omega)$ is the harmonic component and $n(\omega)$ is the noise component. The HNR [11] in decibels then becomes:

$$HNR = 10 \cdot \log_{10} \frac{\int_\omega |h(\omega)|^2}{\int_\omega |n(\omega)|^2},$$

which can be computed as

$$HNR(dB) = 10 \cdot \log_{10} \frac{r'_x(\tau_{max})}{1 - r'_x(\tau_{max})},$$

where $r_x(\tau)$ is the autocorrelation function of the signal, defined as

$$r_x(\tau) = \int x(t)x(t+\tau)dt, \quad r'_x(\tau) = \frac{r_x(\tau)}{r_x(0)}.$$

Although the oscillation generated by the vocal cord is mostly periodic, the speech production system also involves other soft-tissue components that can cause small fluctuations to the signal. As a result, jitter and shimmer are variations in the signal frequency and amplitude [3] caused by irregular vocal fold vibration. An illustration of the jitter and shimmer can be seen in Figure 9.

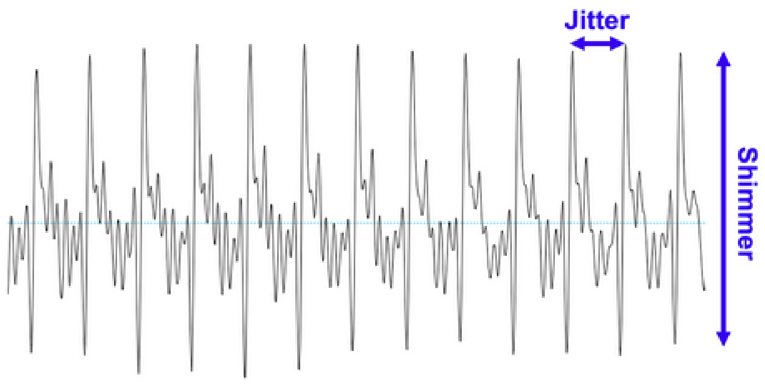


Figure 9: The jitter and shimmer observed in the signal of a speech recording. Taken from [3].

The jitter and shimmer are measurements of the roughness, breathiness, or hoarseness in a speaker's voice, often used to detect pathologies. Many factors can affect these values, such as loudness of voice, gender, language, smoking or alcohol consumption habits. Hence, they could be useful in speech recognition. The Jitter and

shimmer can be measured in several different ways.
Jitter perturbation can be measured with the absolute jitter (jitta) i.e., the absolute perturbation, the local or relative jitter (jitt), the relative average perturbation (rap), and the five points period perturbation (ppq5) [28].
The local absolute jitter is the cycle-to-cycle variation of fundamental frequency, i.e., the average absolute difference between consecutive periods, expressed as:

$$jitta = \frac{1}{N-1}\sum_{i=1}^{N-1} |T_i - T_{i-1}|,$$

where T_i are the extracted F_0 period lengths and N is the number of extracted F_0 periods.
The local relative jitter is the average absolute difference between consecutive periods, divided by the average period, given by:

$$jitt = \frac{\frac{1}{N-1}\sum_{i=1}^{N-1} |T_i - T_{i-1}|}{\frac{1}{N}\sum_{i=1}^{N} T_i}.$$

The relative average perturbation (rap) jitter is the average absolute difference between a period and the average of it and its two neighbours, divided by the average period, given by:

$$rap = \frac{\frac{1}{N-1}\sum_{i=1}^{N-1} |T_i - (\frac{1}{3}\sum_{k=i-1}^{i+1} T_k)|}{\frac{1}{N}\sum_{i=1}^{N} T_i}.$$

The five-point period perturbation quotient (ppq5) jitter is the average absolute difference between a period and the average of it and its four closest neighbours, divided by the average period, given by:

$$ppq5 = \frac{\frac{1}{N-1}\sum_{i=2}^{N-2} |T_i - (\frac{1}{5}\sum_{k=i-2}^{i+2} T_k)|}{\frac{1}{N}\sum_{i=1}^{N} T_i}.$$

Shimmer is a variation of the amplitudes of consecutive periods, measured by subtracting the amplitude of the fundamental frequency period sequence to its neighbor or combinations of its neighbors. Shimmer also has four measurements: the relative shimmer, the local shimmer in a logarithmic domain (ShdB), the three-point Amplitude Perturbation Quotient (apq3), and the five point Amplitude Perturbation Quotient (apq5) [28].
The relative shimmer is the average absolute difference between the amplitudes of consecutive periods, divided by the average amplitude, given by:

$$Shim = \frac{\frac{1}{N-1}\sum_{i=1}^{N-1} |A_i - A_{i+1}|}{\frac{1}{N}\sum_{i=1}^{N} A_i}.$$

Shimmer in dB is the variability of the peak-to-peak amplitude in decibels, is the average absolute base-10 logarithm of the difference between the amplitudes of consecutive periods, multiplied by 20 and given in dB:

$$ShdB = \frac{1}{N-1}\sum_{i=1}^{N-1} |20\log(A_{i+1}/A_i)|.$$

The three-point amplitude perturbation quotient (apq3) shimmer, is the average absolute difference between the amplitude of a period and the average of the amplitudes of its neighbours, divided by the average amplitude, given by:

$$apq3 = \frac{\frac{1}{N-1}\sum_{i=1}^{N-1} |A_i - (\frac{1}{3}\sum_{k=i-1}^{i+1} A_k)|}{\frac{1}{N}\sum_{i=1}^{N} A_i}.$$

The five-point Amplitude Perturbation Quotient (apq5) Shimmer, is the average absolute difference between the amplitude of a period and the average of the amplitudes of it and its four closest neighbours, divided by the average amplitude, given by:

$$apq5 = \frac{\frac{1}{N-1}\sum_{i=2}^{N-2} |A_i - (\frac{1}{5}\sum_{k=i-2}^{i+2} A_k)|}{\frac{1}{N}\sum_{i=1}^{N} A_i}.$$

All these features are computed using the library `parselmouth` [13] in Python, which is a Pythonic interface for the Praat software [6] that is mostly used by linguists.

Other speech characteristics of a person's voice that were tested in the project are speech rate, articulation rate, and the average length of pauses. In phonetics, the two terms, speech rate and articulation rate, were often used interchangeably to imply speech tempo, a measure of the number of speech units of a given type produced within a given amount of time. Nowadays, it is agreed that both speech rate and articulation rate are defined as “the number of output units per unit of time" [30]. However, the former includes pause intervals while the latter excludes them. While the articulation rate only focuses on the pace of speech unit production, the speech rate does take into account a speaker's specific communication style which can include more or less pauses, hesitations, emotional expressions, etc. In this project, speech rate is computed as the number of syllables divided by the time duration of five seconds, and articulation rate is computed as the number of syllables divided by the time duration of five seconds minus the pause duration. The average length of pauses is computed by taking the average of all the pauses' duration in five seconds. For these feature values, the library `parselmouth` in Python is used, just as for the HNR, jitter, and shimmer values.

2.2 Machine Learning

2.2.1 Artificial neural networks

Artificial neural networks (ANNs) consists of many artificial neurons, which loosely model the neurons in a human brain [25]. These neurons are connected between three types of layers: input layer, hidden layer, and output layer as illustrated in Figure 10. Each neuron takes inputs and produces a single output which can be inputs for the neurons in another hidden layer or the final output layer of the network. Examples of the first layer's inputs can be feature vectors of the data, and final output can be the predict class if the task is a classification task.

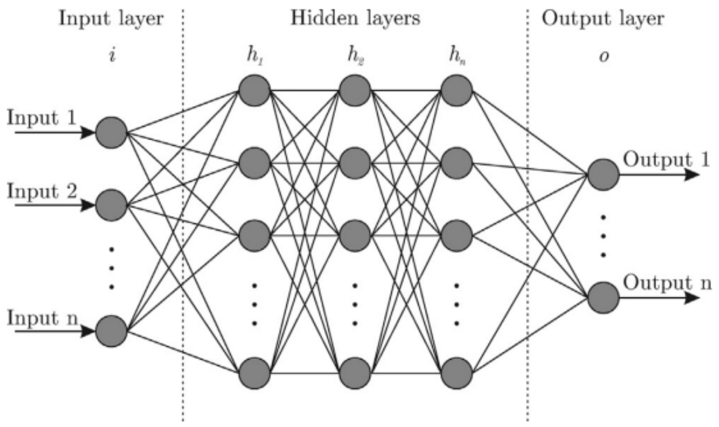


Figure 10: Structure of an artificial neural network. Taken from [7].

The goal when training the network is to minimize a given loss function L. However, due to the non-linearity of the network, the loss function has a non-convex surface, which means that the networks attempt in minimizing the loss function may or may not reach a global minima. The input to the network can be split up into batches, which is how many samples the neural network uses before updating the weights. In this thesis, we used the adaptive moment estimation (Adam) algorithm [15] for the optimization. It is a stochastic gradient descent method that uses the estimation of first- and second-order moments of the gradient to adapt the learning rate for each weight of the neural network. After each batch for the optimization of the loss function, the network goes backwards in each layer to update the weights. These new weights are then used for the next batch where the network goes forward until it reaches the loss function again, and repeat the procedure until it has run through all the batches. That is when the network completes an epoch, and this technique is known as Backpropagation [20]. The number of epochs is the number of times the network goes through all the batches before stopping. To optimize the loss function, sometimes the network requires a large number of epochs, which could result in very long computation time.

2.2.2 Layers

Every layer in the ANN can be constructed differently. Examples of commonly used layers that were tested in this thesis are dense layers, dropout layers, flatten layers, convolutional layers, batch normalization layers, max pooling, average pooling, and global average pooling layers.

Dense layers
The dense layers perform a simple dot product between the inputs x_i and a kernel consisting of weights $w_{j,i}$, resulting in a weighted sum of the inputs. Each hidden layer consists of a pre-specified number of neurons where each neuron then calculates the weighted sum. For example, if the first hidden layer h_1 consists of L neurons, each neuron would perform the following calculations

$$h_1^{(j)} = f\left(\sum_{i=1}^{N} w_{j,i}x_i + b_j\right), \quad j = 1, ..., L \tag{14}$$

where the function $f(\cdot)$ is a pre-specified activation function and b_j are bias terms. These values are then the outputs of the first hidden layer and are used as inputs for the next hidden layer where the procedure of calculating the weighted sum plus some bias term in an activation function is repeated.

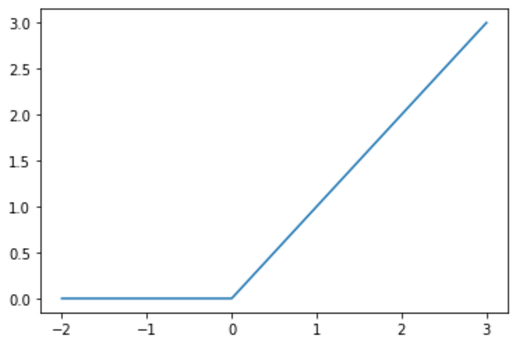
Activation function
The activation function is used to introduce non-linearity in the neural network and determines the output of a neuron given the input. Activation functions used in this project are rectified linear unit (ReLU) and sigmoid. The ReLU function is defined as

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

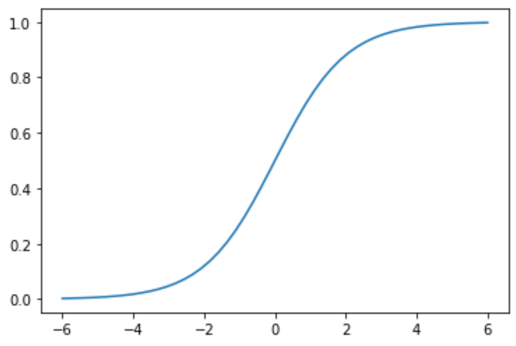
while the sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

An example of how the ReLU function would look like if it received the input $x = [-3, -1, 0, 1, 2, 3]$ can be seen in Figure 11a, where the output would then be $f(x) = [0, 0, 0, 1, 2, 3]$. Moreover, an example of how the sigmoid function would look like for 50 values between -6 and 6 can be seen in Figure 11b



(a) ReLU



(b) Sigmoid

Figure 11: Examples of activation functions used in a neural network

The bias terms b_j are used to shift the activation function to the left or right, which may result in more or less information depending on the activation function.

Dropout layer

The dropout layer randomly chooses some of its inputs and sets them to 0 to help prevent a model from overfitting. The number of inputs to be set to 0, known as $rate$, is commonly chosen somewhere between $20-40\%$, where the inputs that are not set to 0 are scaled up by $1/(1-rate)$ in order to keep the sum over all inputs unchanged.

Flatten layer

The flatten layer takes an input and flattens it into a vector, i.e., if the input is a matrix, the output is then a row vector resulting from concatenating all the rows of the input matrix.

Batch normalization

The batch normalization layer applies a transformation to each batch which maintains the mean output close to 0 and the output standard deviation close to 1. The transformation is done by the following calculations:

$$\gamma \cdot \frac{\text{batch} - \text{mean}(\text{batch})}{\sqrt{\text{Var}(\text{batch}) + \epsilon}} + \beta,$$

where γ, ϵ and β are predetermined coefficients.

Convolutional layers

Convolution is a mathematical operation, denoted as $*$ on two functions (f and h). In deep learning, especially in image processing, the most common type of convolution is the 2D convolution layer. The function $g = (f * h)$ is defined as

$$g(i,j) = \sum_u \sum_v f(i-u, j-v)h(u,v).$$

Here, for example, f can be an input matrix and h is a kernel. A simple example of how the 2D convolution works can be seen in Figure 12, where the kernel is shifted by $stride = (k_1, k_2)$ along each dimension, i.e., it gets shifted k_1 steps to the right until it reaches the end of the matrix, then it gets shifted down k_2 steps and moved all the way to the left of the matrix, then the k_1 shifts to the right is repeated. In this example, the input is a 4×4 matrix (gray), the kernel is chosen as a 3×3 matrix (blue) with $stride = (1, 1)$, and the resulting output is a 2×2 matrix (green). An elementwise multiplication is then performed between the kernel and the elements in the current location of the kernel in the input. The sum of these values constitute an element in the output matrix.

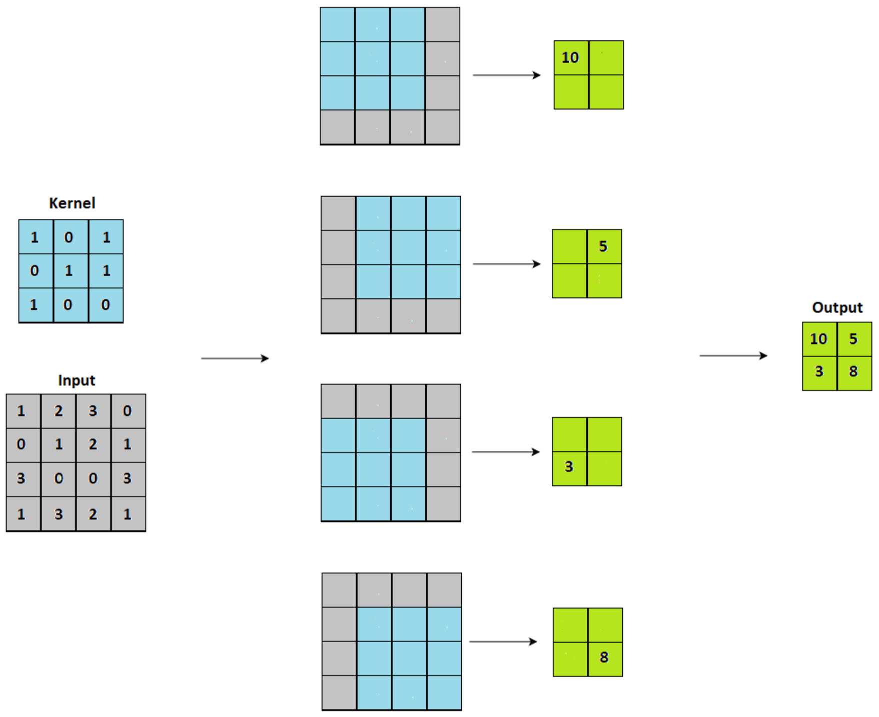


Figure 12: Example of the 2-dimensional convolution operation with an input matrix, a kernel and the resulting output matrix.

The same idea is applied in an 1D convolution layer, where the kernel slides in one dimension instead of two.

Max pooling, average pooling, and global average pooling layers

The max pooling operation downsamples its input by its spatial dimensions by taking the maximum value over a specified window. The average pooling operation is similar to the max pooling but instead of taking the maximum value, it computes the average over the specified window. An example of both max pooling and average pooling can be seen in Figure 13, where the window is then chosen as a 2×2 matrix with $stride = (2, 2)$.

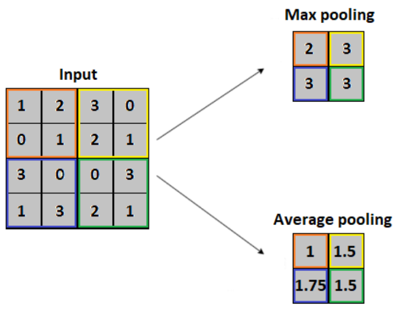


Figure 13: Example of the 2-dimensional maxpooling and average pooling.

The 2D global average pooling would compute the average of the entire input matrix, it can also be a useful way when the spatial dimensions are large and we want to downsample. For example, a tensor with the shape (batch $\times$ rows $\times$ columns $\times$ channels) after global average pooling would become (batch $\times$ channels).

L2 regularization penalty

The idea of the L2 regularization is to make the weights/bias/output of a layer smaller, which may prevent the model from overfitting. The L2 regularization penalty can be used to penalize three different things in a layer. These three penalizers are kernel regularizer, bias regularizer, and activity regularizer. The kernel regularizer applies a penalty on the layer's kernel weights, the bias regularizer applies a penalty on the layer's bias, and lastly the activity regularizer applies a penalty on the layer's output.

2.2.3 Autoencoder

Autoencoder is a type of neural network structure that aims to compress the input data into an encoding, then decompress the encoding into the same shape as the input, such that the output represents the input in a different way. An example of an autoencoder can be seen in Figure 14, where the input is an image of a cat, which then gets compressed by the encoder up until the encoding part, where it then runs through the decoder to decompress the encoding into an image that is as close as possible to the original input.
In this project, the autoencoder was used for feature extraction, however, only the "Encoder" part was implemented to get an encoding, i.e., the "Decoder" part, as seen in Figure 14, of the network was never built or used.

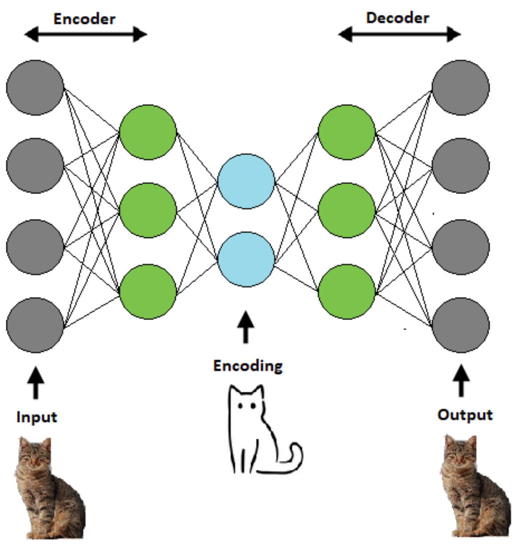


Figure 14: Structure of an autoencoder.

2.2.4 Siamese Neural Network

A Siamese neural network is a class of neural network architectures that consists of two or more identical neural networks, often called sister networks. "Identical" here means that the sister networks have the same configuration, such as layer structure and parameters. These networks take in different inputs and output their individual encodings. These encodings are then fed to a loss function for backpropagation. The loss function comprises of distance/similarity metrics between the encodings in order to minimize the distance between inputs from the same class, and maximizing the distance between inputs from different classes. The final encodings from the network with optimized weights are considered fingerprints of the corresponding inputs. These fingerprints are used to classify which class the original inputs belong to.

In our project, there are two main classes: the speaker and “not the speaker” which includes both their impersonators and other people.

The neural network structure is determined by the loss function. Our project implements three types of loss functions: pair loss, triplet loss, and quadruplet loss.

The pair loss function is defined as

$$L(x_A, x_N) = \max(\alpha - d(x_A, x_N), 0),$$

where x_A is the encoding of an anchor input (the speaker), x_N, encoding of a negative input (not the speaker), α a margin, and d a distance function, which in our case is the Euclidean distance. The loss function aims to maximize the distance between the anchor and the negative encodings. Using this pair loss function, the Siamese neural network's architecture can then be illustrated as in Figure 15.

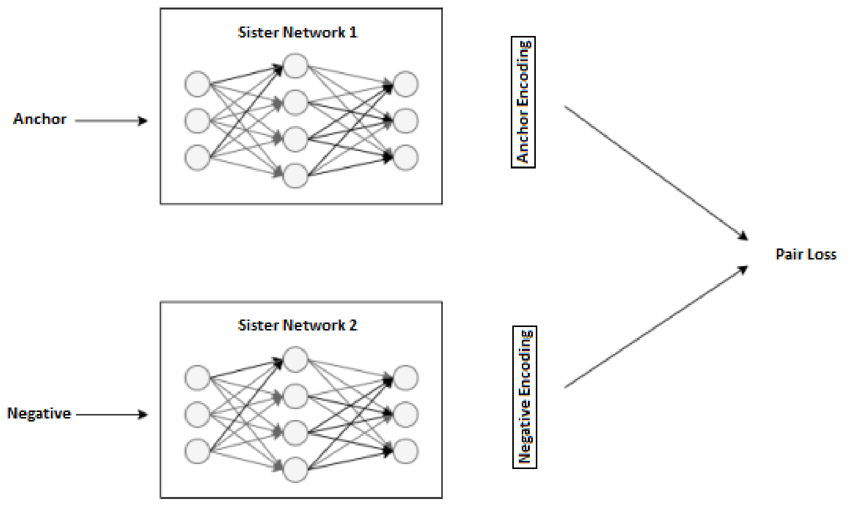


Figure 15: Siamese neural network with pair loss function.

The triplet loss function is defined as

$$L(x_A, x_P, x_N) = \max(d(x_A, x_P) - d(x_A, x_N) + \alpha, 0),$$

where a new variable x_P is introduced, which is the encoding of a positive input (the speaker's recording at another time). The triplet loss function aims to maximize the distance between the anchor and negative encodings while at the same time, minimizing the distance between the anchor and positive encodings. Using this triplet loss function, the Siamese neural network's architecture can be illustrated as in Figure 16.

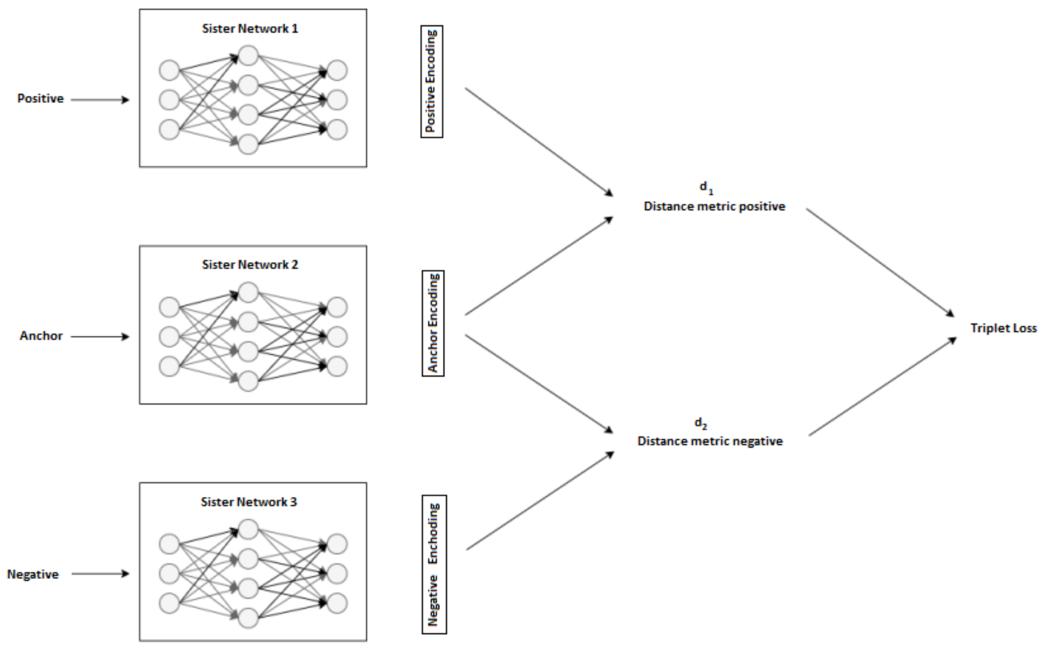


Figure 16: Siamese neural network with triplet loss function.

The quadruplet loss function [9] is defined as

$$L(x_A, x_P, x_{N_1}, x_{N_2}) = \max(d(x_A, x_P) - d(x_A, x_{N_1}) + \alpha_1, 0) \\ + \max(d(x_A, x_P) - d(x_{N_1}, x_{N_2}) + \alpha_2, 0),$$

where two negative inputs are used, resulting in variables x_{N_1} and x_{N_2}, which are encodings of the two different negative inputs, and also α_1 and α_2 being the margins. The quadruplet loss function aims to maximize the distance between the anchor and negative encodings while at the same time, minimizing the distance between the anchor and positive encodings as well as maximizing the distance between different negative encodings. Using this quadruplet loss function, the Siamese neural network's architecture can then be illustrated as in Figure 17.

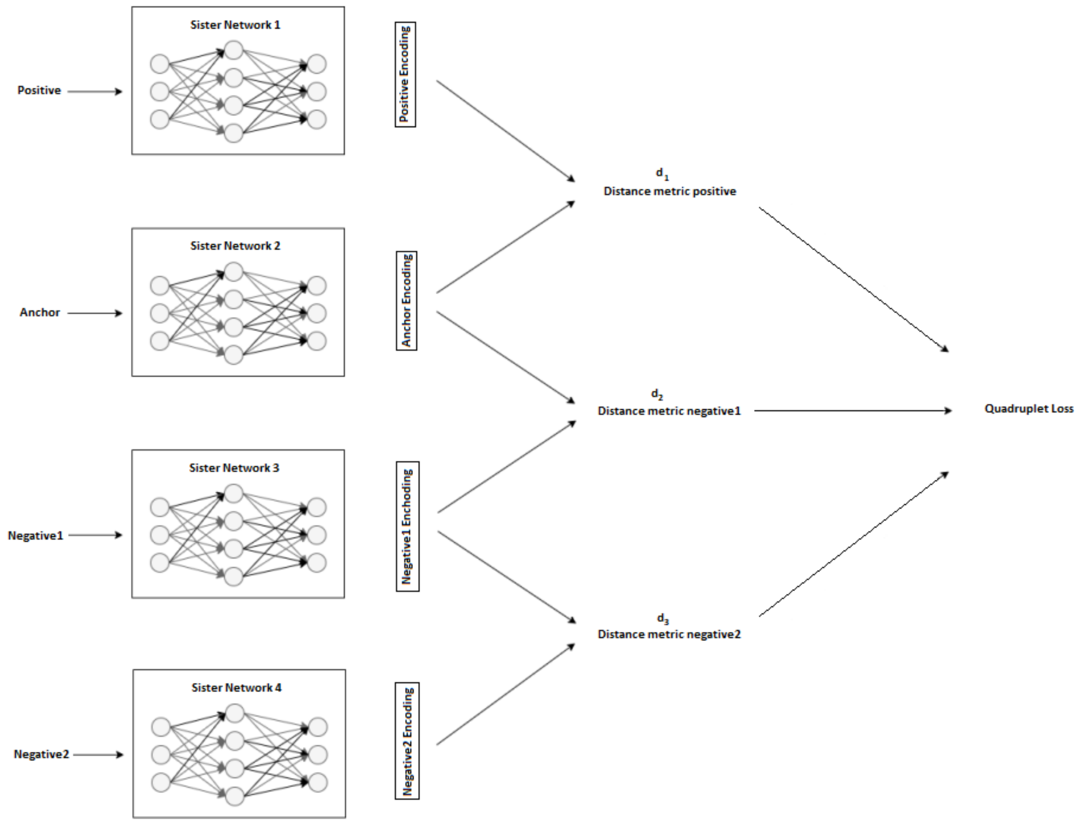


Figure 17: Siamese neural network with quadruplet loss function.[9]

2.2.5 Classifiers

After optimizing the Siamese neural network for the speaker and not-the-speaker classes, the outputs of this network, i.e., encodings of the feature vectors for each voiced segment, are then used as fingerprints for each voiced segment. Using these fingerprints, a classifier can be trained to identify the speaker. The two classifiers that are chosen to be experimented in the project are k-nearest neighbor (kNN) and random forest classifiers.

k-Nearest Neighbour Classifer

kNN is a type of supervised learning algorithm that can be used for classification. The kNN predicts the class of a test sample by calculating the distance from the test sample to all the training samples, then select k number of points that are closest to the test point. The class that occurs the most frequently in these k points is chosen as the predicted class for the test sample. This is also known as the brute force algorithm. [2]

This project uses the ball tree algorithm which has already been implemented in Python's `scikit-learn`. The idea of the ball-tree algorithm is to partition the data into a series of nesting hyper-spheres, each defined by a centroid and a radius. A test sample point must end up in one of these nested balls. Points within this nested ball are expected to be closest to the test data point. From here, instead of computing the distance from the test point to all the training points, only the distances from the test point to each point in the ball are computed. This makes the algorithm much more efficient when the dimension is large. Thereafter, the predicted class is determined by the k-nearest neighbors.

Random forest classifier

Random forest is a supervised learning algorithm that consists of a number of decision trees that operates as an ensemble. Each individual decision tree outputs a predicted class, and the class that is most frequently predicted becomes the predicted class of the whole model.

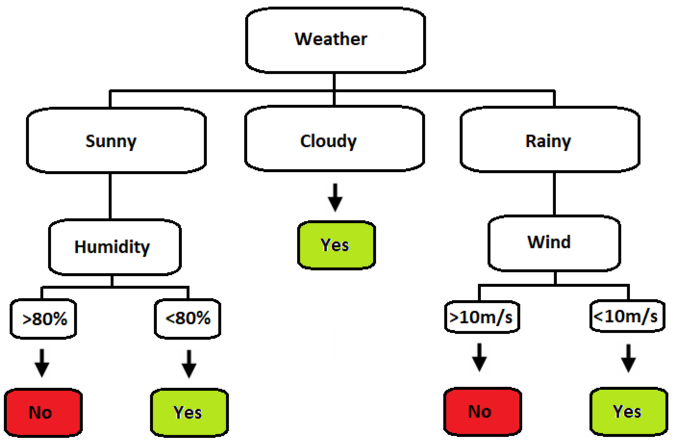


Figure 18: Example of a decision tree.

A decision tree is a machine learning algorithm that has a tree-like structure. A decision tree typically starts with a root node, which then branches into different outcomes, each of which can lead to smaller branches of different possibilities, and so on. The size of the tree depends on the number of features the data has, i.e.,

the more features, the larger the tree. It is up to the user to decide how many features to include, or to use pruning to reduce the size of the tree, in order to avoid overfitting and high computational cost.
For example, given some weather data and exercising habits of some people, a decision tree can be used to predict, whether a person is going to exercise outdoors or not. An illustration of this can be seen in Figure 18. Based on the weather condition, if it is cloudy, or sunny with humidity below 80%, or windy with wind speed below 10m/s, then a person is predicted to exercise outdoors. If it is sunny with humidity above 80%, or windy with wind speed above 10m/s, then a person is predicted to not exercise outdoors.

3 Method

3.1 Data processing and feature extraction

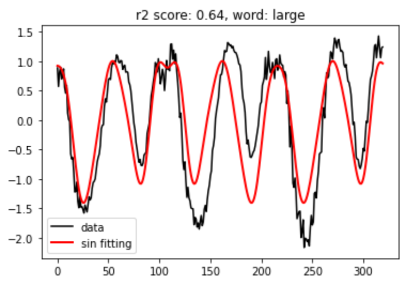
3.1.1 Data collection

The data used in this project are clips of prominent political figures such as Donald Trump and Barack Obama giving speeches and their respective impersonators, as well as clips of other individuals speaking. These clips are then converted to audio files, *wav* format, sampling rate 16000, PCM 16-bit Mono. These audio files are used for both our training, validation, and test data. Specifically, for each speaker, for the training data, for the anchor and for the positive data sets, a recording or a combination of recordings of total length around ten minutes are used, and for the negative data sets, recordings of different people that make up a total length of also about ten minutes are used.

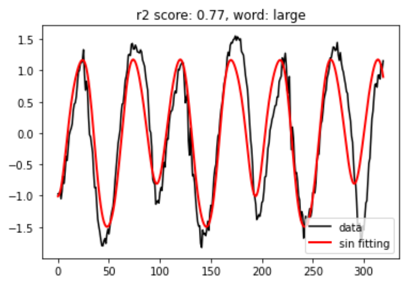
3.1.2 Data processing

Before starting the sinusoidal fitting on the recordings, all the silence was removed in order to avoid unnecessary computational power by trying to fit a sinusoid to a straight line. In order to remove the silence from the recordings, the open source speech recognition toolkit, Vosk [8] was used. There are probably easier ways to remove the silence from an audio recording, but Vosk allowed us to also see what word the person speaking was saying as well as the time period for the word. This way, specific words that, when plotting and inspecting them, did not have a periodic structure, could also be filtered out. These words are often function words such as "to", "has", "have", or "were", that are often not emphasized in a sentence. The last step before the sinusoidal fitting started was to ensure that the segment was not the onset or the end of a consonant, which is often silence or has a sound wave with very small amplitudes. This was done by calculating the sum of the absolute amplitudes of the signal divided by the sampling rate. As expected, the segments that included consonants had values that were much smaller than those of vowel segments. By looking at multiple plots of segments and their corresponding absolute relative amplitudes, a threshold of 5 was deemed reasonable. Any segment with this value larger than the threshold was passed into the sinusoidal fitting, while the others were was discarded. Of these segments that passed the threshold, only those with a sine fitting R^2 score larger than 0.5 were included in the final data set. The threshold 0.5 was chosen after having looked at many different sine fittings of segments, where most of the segments that lacked a periodic structure had an R^2 score below 0.5.

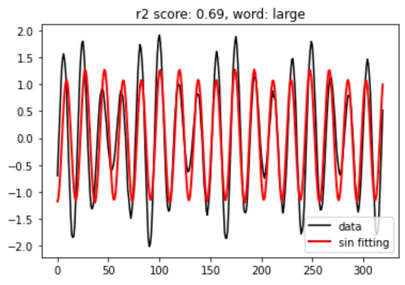
An example where the sine-fitting was used in order to find periodic structures can be seen in Figures 19 and 20, where the segments that are parts of the big segment that contain the word "large" are fitted. The sine-fitting scheme managed to find 8 segments of 20 ms where the R^2-score was larger than the threshold of 0.5. Figures 19a and 19b are 20-millisecond segments of the phoneme represented by the letter "l" and Figures 19c and 19d are 20-millisecond segments of the phoneme represented by the letter "a".



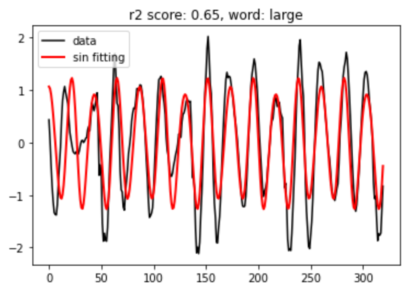
(a) Periodic structure in l



(b) Periodic structure in l



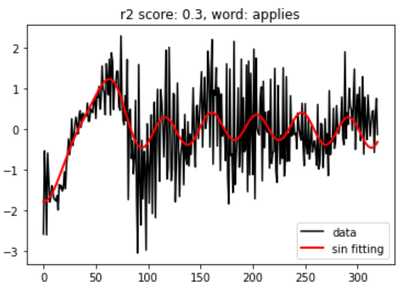
(c) Periodic structure in a



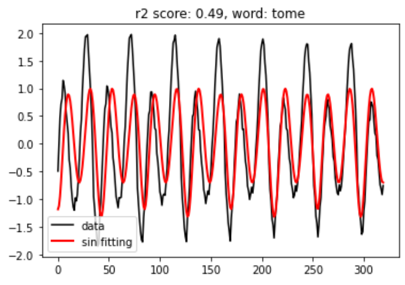
(d) Periodic structure in a

Figure 19: Sine fit of saved segments of two different phonemes.

Thus, our data set contains millisecond segments of both vowels and also consonants that have periodic structures.
Figure 20 shows examples of segments that are discarded. It can be either a segment of a consonant with no periodic structures, as in Figure 20a with an R^2 score well below the threshold, or a segment with a quite nice periodic structure as in Figure 20b with an R^2 score that is very close to the threshold.



(a) Consonant



(b) Phoneme in tome

Figure 20: Sine fit of discarded segments.

It is understood that by using R^2 as measure and setting the threshold to be 0.5, some data of interest are thrown away. However, given the amount of data and the computational cost, this was deemed an acceptable trade-off.

3.1.3 Feature extraction

From the data sets that contain 20-ms and 40-ms segments with periodic structures, the features that were used were MFCCs, formants F1, F2, F3, F4, formant bandwidths for these first four formants, and formant ratios of F2 and F1. For MFCCs, Python's built-in `mfcc` function in the `python_speech_features` library was used, for the formants, bandwiths and ratios, Python's `lpc` function in the `librosa` library was used to get the model polynomial. The formant frequencies are then obtained by finding the roots of the prediction polynomial. The algorithm for formant estimation with LPC coefficients was adapted from [19].

From the data set containing just the original recordings, these recordings are segmented into five-second segments, from which the spectrograms are computed. To compute the spectrograms, different parameters such as different windows, window lengths, standard or multi-taper and compare the performances were tested. For example, when using the Tukey windows, eight overlapping windows were used, whereas with the sine windows eight different windows were chosen. However, using the overlapping Tukey windows seemed to perform better and was thus the method that was used in this project. The length of the Tukey windows were chosen such that the spectrogram was split into 11 time segments. The spectrograms were also cut to only include frequencies between 0-1200Hz, as frequencies larger than 1200Hz had amplitudes close to 0 and did not seem to contain much information.

For the 20-ms and 40-ms data sets, the MFCCs, formants, bandwiths and formant ratios for each segment are computed. These values are stored as row vectors, where 50 segments of the 20-ms feature vectors were stacked into one feature matrix (which would be one sample), and 10 segments of the 40-ms feature vectors were stacked into another feature matrix. When computing the MFCCs on the 40-ms data, they are segmented into two 15-ms segments and one 10-ms segment resulting in a 3×12 matrix of MFCCs for each 40-ms segment. Thus, 10 stacked segments of MFCCs, formants, bandwidths and formant ratios of the 40-ms data gives a matrix of dimensions 30×17, whereas the 50 stacked 20-ms segments resulted in a matrix of dimensions 50×17. For the five-second segment data set, the spectrograms are computed and only the significant parts are kept as specified above. As a result, these give a feature matrix of size 600x11 representing a sample.

From these two types of inputs, first a bi-stream neural network architecture for the sister network was experimented. Each input is fed to a network, then the outputs are concatenated and fed to another Dense Layer before returning the final encodings. An example of a network that was experimented can be seen in Figure 35, where input 1 (right branch) corresponds to the spectrograms from five-second segment data and input 2 corresponds to the stacked feature vectors from the 20-ms segment data. Another approach is to concatenate the feature matrices from all three data sets into a single matrix, which is then used as a single input for the sister network.

3.2 Neural Network Model and Classifier

To build the neural network models, the free open-source software library `TensorFlow` and the API `Keras` in Python were used. However, there are many different parameters that one needs to decide in order to start training a model. These include how many epochs one wants the model to have, i.e., how many iterations of the training data set one wants the model to train on. Other parameters can be the learning rate which determines how big of a step size one wants the optimizer to take in each iteration, as well as what layers to use, the size of each layer, the dimension of the kernels, and the batch size which determines the number of data points used in each iteration in the training.

To find out which combinations of the above parameters that works well, the `HyperParameter` function from the `keras_tuner.engine.hyperparameters` [22] library in Python was used. The Keras Tuner helps to pick optimal set of hyperparameters for neural networks created with `TensorFlow`. One chooses the architecture of the network, i.e., what layers to use, then one decides what different dimensions of layers that one wants the `HyperParameter` to try, as well as different loss functions, different margins for the loss functions, learning rate, stride dimensions etc. It then goes through all the different combinations of pre-specified hyperparameters and trains the network and evaluates the performance by an optional metric. In our case, the performance is evaluated by the value of the validation loss. The size of the dense layers, kernel dimensions, filters, learning rate, loss function, margins, batch size, and stride dimensions that resulted in giving the smallest validation loss was then used as our model. Since our input matrix of features for the neural networks was not a square matrix, a decision to also let the `HyperParameter` try non-square kernels for the Conv2D layers was made, which have been shown to perform well for some cases [17].

Once the networks were tuned, the decision for the parameters of the classifiers, such as the number of neighbors k in the kNN, and the number of trees in the random forest classifier, had to be made. After trying different values of k, $k = 1$ and $k = 3$ were chosen to be experimented as they gave the most preferable results, whereas the random forest classifier seemed to perform best using 100 trees with the mean squared error as the metric measuring the quality of a split.

3.3 Model performance evaluation

Choice of negative recordings

The choice of which recordings to use as the negative (not the speaker) was also an important aspect, since if the speaker and the negative data already have very different voices, then the resulting model might not need to be very good. This means that the dissimilarities may result in having a large Euclidean distance of the encodings even before training, the networks would therefore not have to optimize any weights. Thus, using only recordings of females might not be a good idea when the goal is to classify a male with a deep voice, since their voices might already be very dissimilar. In order to achieve a good model, the goal is to make the distance of people that sound similar to the anchor as large as possible. When testing the trained network, with a bad choice of negative, on someone that sounds similar to the anchor, it very often resulted in classifying it incorrectly. Therefore, many different SNN's using pair loss were investigated, by training them with the anchor and one of the random people at a time as negative. This allowed us to see the individual distance between the random person and the anchor, as well as how hard it was for the network to separate them. The people that resulted in giving distances of both training and test data close to 0 after training, were put as negative. Since they either sound similar to the anchor, or the network is simply having a hard time optimizing for the corresponding person.

In order to get an overall good separation between the anchor and the recordings that were deemed to be good negatives, the good negatives were concatenated into a single negative vector. However, to ensure that the recordings that were easy for the network to separate from the anchor stayed that way, some of the easier recordings were added to the big negative vector.

Receiver Operating Characteristic

A receiver operating characteristic (ROC) curve is a graph showing the performance of a classification model by plotting the true positive rate against the false positive rate. True positive (TP) is a test result that correctly indicates the presence of a characteristic, and false positive (FP) is a test result that incorrectly indicates that. True negative (TN) is a test result that correctly indicates the absence of a characteristic, and false negative (FN) is a test result that incorrectly indicates that. Using these definitions, the true positive rate (TPR) is defined as

$$TPR = \frac{TP}{TP + FN},$$

and the false positive rate (FPR) is defined as

$$FPR = \frac{FP}{FP + TN}.$$

A ROC curve plots TPR vs. FPR at different classification thresholds. A perfect classifier will have TPR=1 and FPR=0, where all the tests are correctly labeled. The higher the curve reaching the left corner point (0,1), the better the classification model, as illustrated in Figure 21 below.

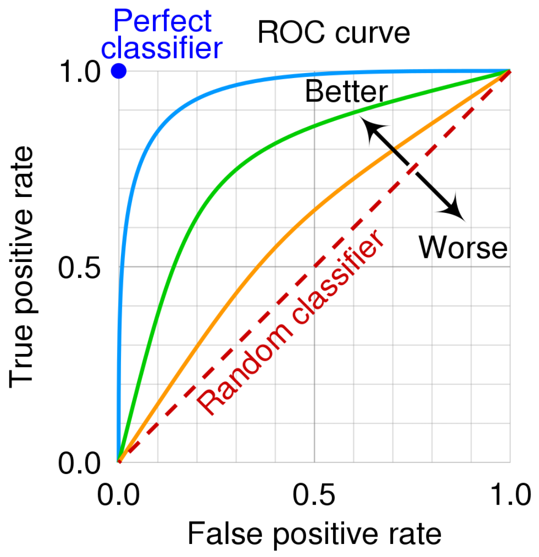
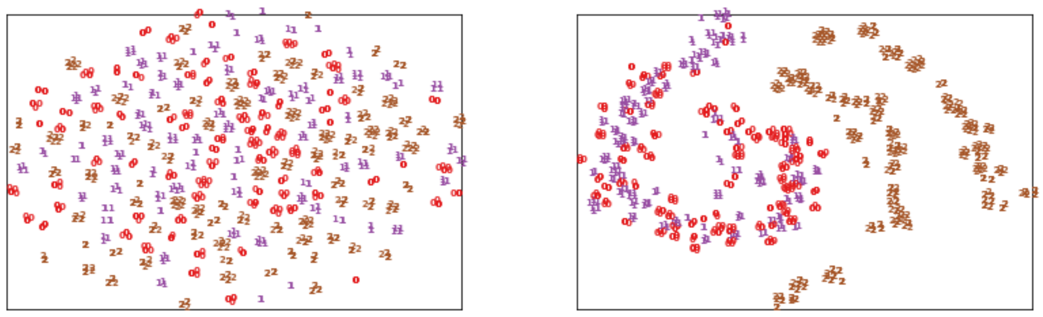


Figure 21: ROC curve, from Wikipedia.

From the ROC curve, there is another performance measurement of a classification model, which is the area under the ROC curve (AUC). The AUC can have values between 0 and 1, with 1 indicating the perfect classifier. At 0.5, AUC implies a random classifier and at 0, it means that the model predicts all the test cases wrong.

T-distributed Stochastic Neighbor Embedding

The idea behind the t-distributed Stochastic Neighbor Embedding (t-SNE) is to visualize high-dimensional data by giving each data point a location in a two-dimensional map [18]. It was used as a visualization tool of our encodings (fingerprints) produced by the SNN's, in order to confirm that the network was doing what it was supposed to do. This allowed us to see if the Euclidean distance between the anchor encoding and positive encoding became smaller, as well as the Euclidean distance between the anchor encoding and negative encoding became larger. This can be seen in Figure 22, where in (a) the encodings produced by an untrained SNN using triplet loss can be seen, and in (b) the results of encodings produced by the same network but after training is shown. The red number "0" in the plot corresponds to the anchor, the purple "1" corresponds to the positive, and the brown "2" corresponds to the negative. The trained model does indeed seem to



(a) Untrained Siamese triplet (b) Trained Siamese triplet

Figure 22: t-SNE plot of encodings where the red "0" corresponds to the anchor encoding, the purple "1" correspond to the positive encoding, and the brown "2" correspond to the negative encoding.

make the distance between the anchor and positive smaller, while simultaneously making the distance between the anchor and negative larger.

Test data and Accuracy

In order to test the performance of the model, two different types of test data were used. Before training the model, the data are shuffled and split up into 70% training, 10% validation, and 20% test. However, even though the trained model has never seen the 20% test data, they are segments of the same recording that the model was trained on. This type of test data is what was used to generate the ROC curves and AUC scores for evaluating the model. However, for the accuracy tables reported in the Results section later, completely new recordings were used as the second type of test data. These recordings consisted of the true speaker (anchor) at some other time point, three random people (both male and female), and the impersonators.

The new test data, i.e., the second type, were also put into the data processing step before testing them in the trained model. Each recording of these individuals can generate several sample tests (depending on the length of the data as well as how many 20- and 40-ms segments that were found), the accuracy of the model on each recording is measured by how many test samples were correctly classified divided by the total number of samples from that recording.

4 Results

After testing many different combinations of features as well as network architectures, the features that performed significantly better than others were the concatenated MFCCs, formants, and formant ratios between second and first formants, all on the 20-ms and 40-ms segments, as well as the spectrogram on the five-second segments. The structure of the concatenated input feature matrix can be seen in Figure 23, where some zero paddings were used for the spectrogram in order to get the dimensions of the matrix to fit.

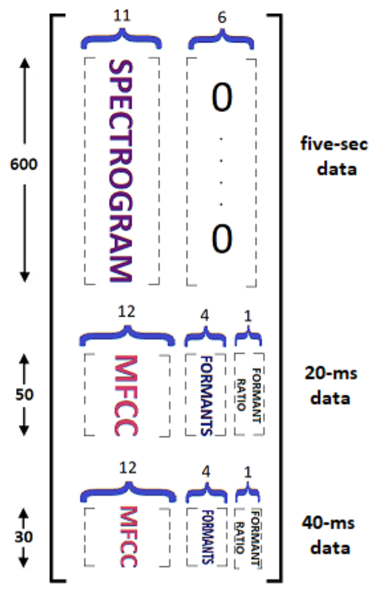


Figure 23: Input feature matrix for the SNN's.

These features together with the network architecture presented in Figure 24 were then used to compare the performance of the three different loss functions as well as when having different people in the negative training data. The 2-dimensional convolution layer had kernel dimensions 10×2 with $stride = (1, 1)$ and 48 output filters, the kernel and bias also had a L2 regularization penalty equal to 0.01. The max pooling layer had window dimensions also equal to 10×2 (with strides also equal to $(10, 2)$). After taking the global average, the outputs were sent through a standard dense layer, from which the outputs were normalized in the batch normalization layer. After the normalization, the outputs were sent into another dense layer, which also consisted of a L2 regularization penalty equal to 0.1, but this time on the activation function. Lastly, the outputs were then sent into the last dense layer, which then returned the encoding. The 2-dimensional convolution layer and the first dense layer before the batch normalization had ReLU activation function, whereas the second last dense layer after the batch normalization had a sigmoid activation function. The rest of the network did not use activation functions.

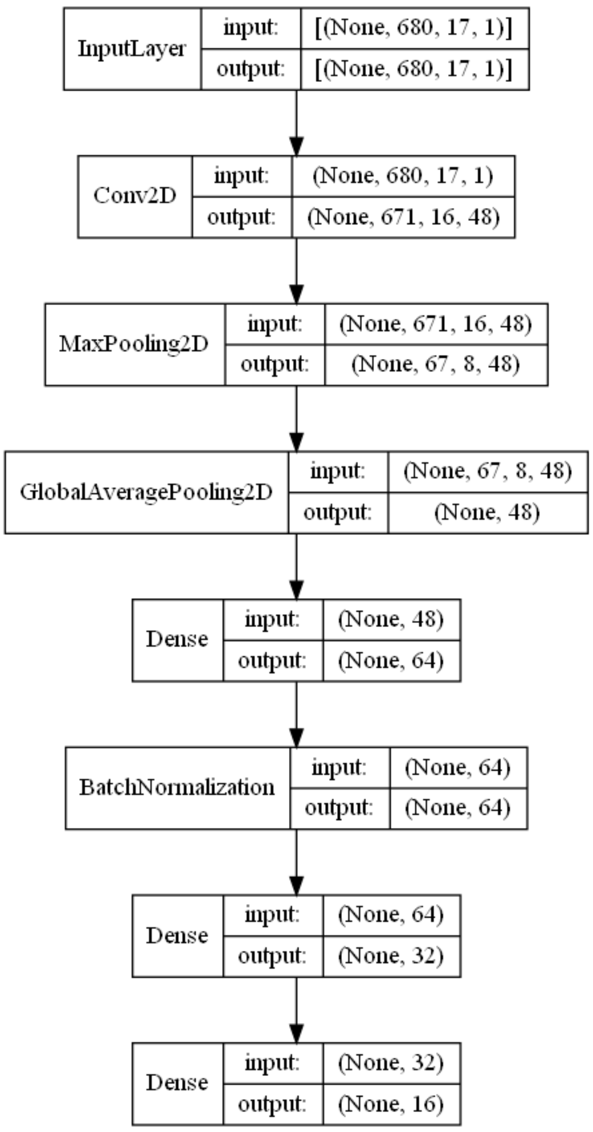


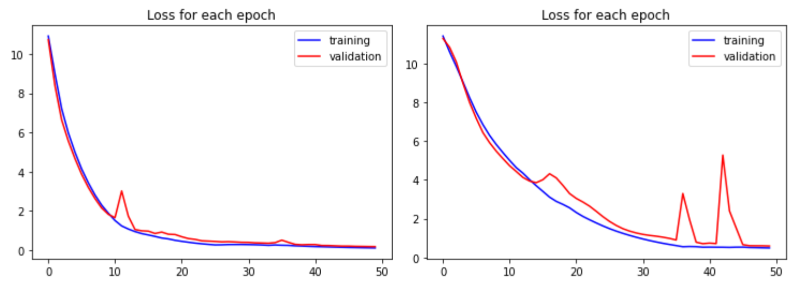
Figure 24: Sister network architecture.

Since these features and network architecture were superior to all of the other feature combinations and network architectures that were tested, this setup was used in networks using the three different loss functions, i.e., pair loss, triplet loss, and quadruplet loss for comparison. The margin α in the pair and triplet loss function did not have a significant effect, and was thus set to $\alpha = 0.9$. This was also the case for the quadruplet loss, as long as $\alpha_2 < \alpha_1$ was satisfied, it did not have a significant effect on the result. Thus, they were set to $\alpha_1 = 0.9$ and $\alpha_2 = 0.6$. These loss functions were compared for two different true speakers, i.e., Donald Trump as the anchor and Barack Obama as the anchor. For Donald Trump, for the network using the pair loss function, the difference in performance when using random people in the negative training data, compared to carefully chosen people in the negative training data was investigated, as this turned out to be an important factor in how well the networks performed.

4.1 Speaker Donald Trump

Pair loss

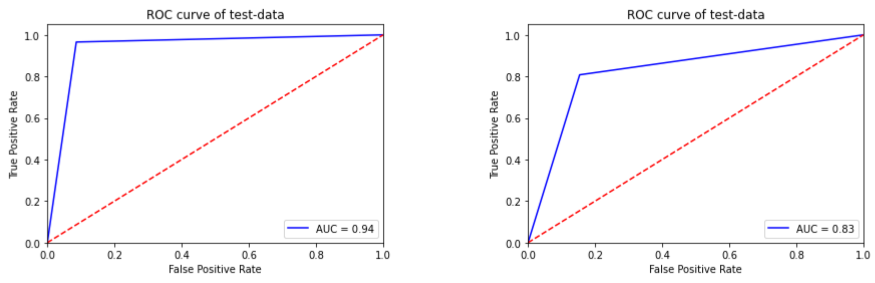
The performance of the network corresponding to the pair loss function is shown by comparing the network with a negative set carefully chosen, as explained in section 3.3 (Choice of negative recordings), and the one with a randomly chosen negative set. How the loss function behaved for each epoch can be seen in Figure 25.



(a) Chosen negatives **(b)** Random negatives

Figure 25: Training loss (blue) and validation loss (red) after each epoch.

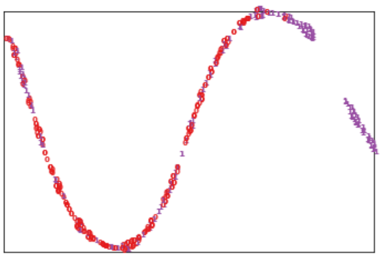
For both cases, the loss function on the training data decreases somewhat steadily. However, with the chosen negatives, it decreases much faster, already starting to stabilize and only decreasing slightly after epoch 20. Meanwhile, for the random negatives, it starts to look like it is stabilizing after epoch 40. On the validation data, however, large fluctuations around epochs 40 to 50 are observed in the network using random negatives while much more stable behavior towards larger epochs is seen in the chosen negatives. It can be concluded that using carefully chosen data for the negative set is likely to result in a more stable, better-performing model. This is confirmed by looking at the ROC curves for the test data from the two cases in Figure 26.



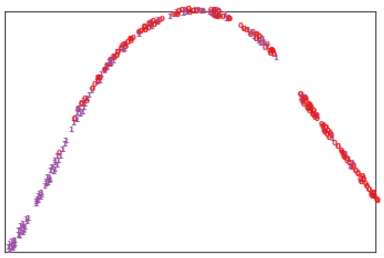
(a) Chosen negatives **(b)** Random negatives

Figure 26: ROC curves and AUC scores of the 20% test data, (a) AUC =0.94, (b) AUC = 0.83.

The ROC curves show that using the randomly chosen data for the negative set, in this case, results in a model with a lower AUC score. Using the carefully chosen data, on the other hand, results in a AUC score of 0.94, which implies a quite



(a) Chosen negatives

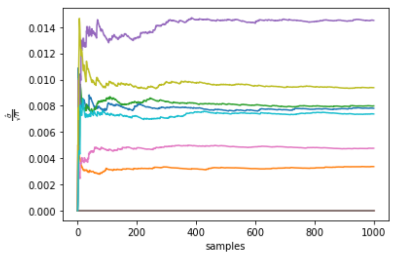


(b) Random negatives

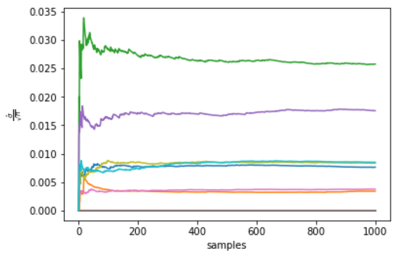
Figure 27: tSNE plots of the training data encodings of the trained model.

good model. Moreover, tSNE plots are also used to show how well the data for different classes are separated. As can be seen in Figure 27, in both cases, the true speaker (red zeros) are clustered together more to the left, and the other speakers or negative data (purple ones) to the right. There are several purple ones that are clustered with red zeros but both model could separate the majority of the data.

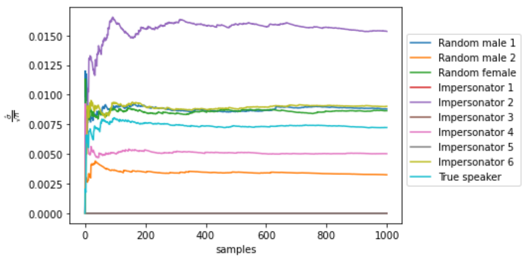
Since it took quite long to train a network, it was decided that 100 different trained networks with chosen negative data were generated, and the accuracies on the test data (type 2) were computed. From these 100 accuracies from each network, the bootstrapping with replacement 1000 times were used to generate more samples for estimating the mean accuracies and confidence intervals. The standard deviations of the mean accuracies were observed to determine the number of samples required for them to start converging. As can be seen from Figure 28, already at 100 up to 200 samples, the standard deviations have become stable.



(a) kNN $k = 1$



(b) kNN $k = 3$



(c) Random Forest

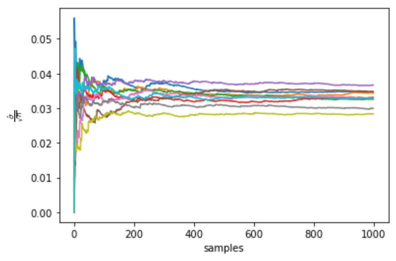
Figure 28: Standard deviations of 1000 bootstrapped (with replacement) samples of the means of the accuracies with chosen negative data, using kNN $k = 1$ in (a), kNN $k = 3$ in (b), and random forest in (c).

These 1000 bootstrapped samples were then used to estimate the average accuracies as well as their confidence intervals, which are shown in Table 1.

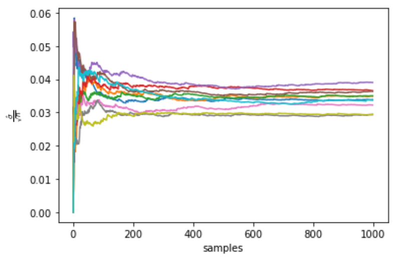
Speakers	kNN $k = 1$		kNN $k = 3$		Random Forest	
	CI	Mean	CI	Mean	CI	Mean
Random male 1	[0.97, 0.99]	**0.98**	[0.96, 1]	**0.98**	[0.96, 1]	**0.98**
Random male 2	[0.99, 1]	**1**	[0.99, 1]	**1**	[0.99, 1]	**1**
Random female	[0.96, 1]	**0.98**	[0.72, 0.82]	0.77	[0.96, 1]	**0.98**
Impersonator 1	[1, 1]	**1**	[1, 1]	**1**	[1, 1]	**1**
Impersonator 2	[0.54, 0.60]	**0.57**	[0.52, 0.60]	0.56	[0.54, 0.60]	**0.57**
Impersonator 3	[1, 1]	**1**	[1, 1]	**1**	[1, 1]	**1**
Impersonator 4	[0.95, 0.97]	0.96	[0.97, 0.99]	**0.98**	[0.95, 0.97]	0.96
Impersonator 5	[1, 1]	**1**	[1, 1]	**1**	[1, 1]	**1**
Impersonator 6	[0.79, 0.83]	0.81	[0.81, 0.85]	**0.83**	[0.79, 0.83]	0.81
True speaker	[0.75, 0.77]	0.76	[0.72, 0.76]	0.74	[0.76, 0.78]	**0.77**
Overall accuracy	**0.907**		0.886		**0.907**	

Table 1: Bootstrapped means and confidence intervals of the 100 accuracies (1000 bootstrapped samples), using pair loss with chosen negative data, of random people speaking as well as impersonators and different recordings of the true speaker (Donald Trump).

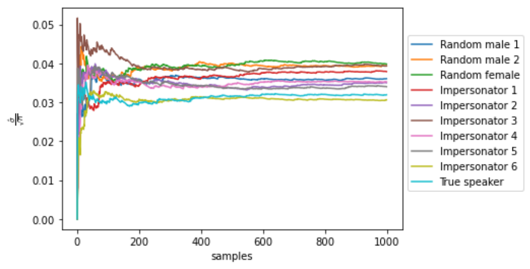
As can be seen from Table 1, the accuracies by all three classifiers do not differ significantly. Only looking at the overall mean, the kNN with $k = 1$ and random forest classifiers seem to perform better than kNN with $k = 3$. For all three classifiers, the average accuracies were exactly equal for five out of ten recordings that were used for testing. Both random forest and kNN with $k = 1$ had exactly the same results, except for the true speaker where random forest was just 1% better. Impersonator 2 was the hardest one for the model to predict, as all three classifiers could only get around 56%, 57% accuracy. Moreover, the best classifier for the true speaker recordings, random forest, could only get 77% accuracy while the other two classifiers were only behind by 1% and 3%. However, compared to the other tests, the accuracies for true speaker were significantly lower, for all three classifiers. This can be observed in Table 1, where all the other tests had the accuracy of around 96% to 100%, except for impersonator 6's , which was about 82%.
In order to show the impact of the choice of recordings in the negative data set, 100 different trained networks using randomly chosen recordings in the negative data were also generated. From these 100 networks, again 1000 bootstrapped samples were computed and the resulting standard deviations were plotted in Figure 29.



(a) kNN $k = 1$



(b) kNN $k = 3$



(c) Random forest

Figure 29: Standard deviations of 1000 bootstrapped (with replacement) samples of the means of the accuracies with random negative data, using kNN $k = 1$ in (a), kNN $k = 3$ in (b), and random forest in (c).

As can be seen from Figure 29, the standard deviations do converge and they start converging after around 200 samples. From the bootstrapped samples, the mean accuracies and their confidence intervals were then computed and presented in Table 2.

Speakers	kNN $k = 1$		kNN $k = 3$		Random Forest	
	CI	Mean	CI	Mean	CI	Mean
Random male 1	[0.72, 0.86]	0.79	[0.78, 0.92]	**0.85**	[0.72, 0.86]	0.79
Random male 2	[0.79, 0.93]	**0.86**	[0.78, 0.92]	0.85	[0.73, 0.89]	0.81
Random female	[0.80, 0.92]	**0.86**	[0.64, 0.78]	0.71	[0.71, 0.87]	0.79
Impersonator 1	[0.80, 0.94]	**0.87**	[0.78, 0.92]	0.85	[0.73, 0.89]	0.81
Impersonator 2	[0.12, 0.26]	0.19	[0.14, 0.28]	**0.21**	[0.11, 0.25]	0.18
Impersonator 3	[0.79, 0.93]	**0.86**	[0.78, 0.92]	0.85	[0.72, 0.88]	0.80
Impersonator 4	[0.76, 0.90]	0.83	[0.79, 0.91]	**0.85**	[0.74, 0.88]	0.81
Impersonator 5	[0.78, 0.90]	0.84	[0.79, 0.91]	**0.85**	[0.75, 0.89]	0.82
Impersonator 6	[0.69, 0.79]	0.74	[0.74, 0.86]	**0.80**	[0.65, 0.77]	0.71
True speaker	[0.62, 0.74]	0.68	[0.58, 0.70]	0.64	[0.64, 0.76]	**0.70**
Overall accuracy	**0.752**		0.746		0.732	

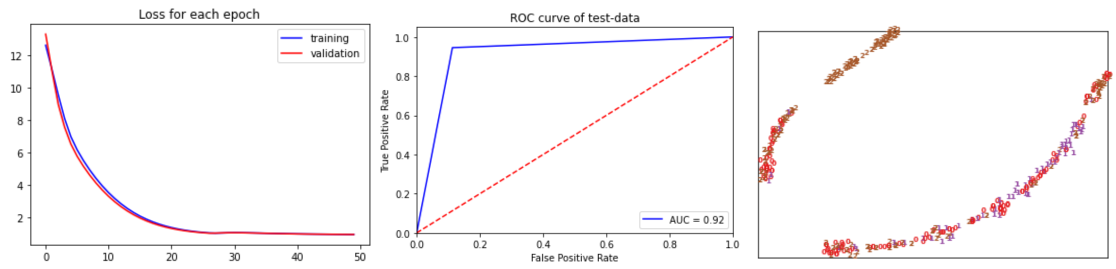
Table 2: Bootstrapped means and confidence intervals of the 100 accuracies (1000 bootstrapped samples), using pair loss with random negative data, of random people speaking as well as impersonators and different recordings of the true speaker (Donald Trump).

Judging from the overall accuracies, the networks using randomly chosen negative data did not perform as well as the ones using carefully chosen negative data. There was roughly 15% difference in the overall accuracies when using different negative data. The best accuracy for impersonator 2, which had been the most difficult one for the networks even when using carefully chosen negative data, was only 21% here, compared to 57% before. Apart from the accuracy for the true speaker, which was still 70% with the best classifier, the accuracies for all the other speakers including random people as well as the impersonators were lower by 5% to 10 % compared to the networks using carefully chosen negative data, which resulted in the lower overall accuracies.

Naturally, a reasonable conclusion would be that the choice of which recordings to be included in the negative data did make a difference in the performance of the networks. Therefore, for the networks using triplet loss and quadruplet loss functions, only carefully chosen recordings in the negative data were used.

Triplet loss

The network's attempt in minimizing the loss function for each epoch, the ROC curve and AUC score, and the tSNE plot of the training data encodings when using the triplet loss can be seen in Figure 30, respectively.



(a) Loss **(b)** AUC **(c)** tSNE

Figure 30: Training loss (blue) and validation loss (red) after each epoch in (a), ROC curve and AUC score of the 20% test data with AUC = 0.92 in (b), and tSNE plot of the training data encodings of the trained model in (c).

In Figure 30 (a), it can be noted that the loss seems to decrease steadily for both the training and validation set and starts to stabilize at around 30 epochs. The ROC curve together with its AUC score of $AUC = 0.92$ for the 20% test data can be seen in (b), which also indicate a decent model. Looking at the tSNE plot of the training data encodings in (c), it does seem that the network is trying to make the distance between the anchor (red zeros) and positive (purple ones) smaller, while simultaneously attempting to push away the negative (brown twos), i.e., making the distance between the anchor and negatives larger.

The standard deviations of the 1000 bootstrapped samples of the mean of the accuracies of the speakers in Table 3 and how they converged can be seen in Figure 36 in the Appendix. The standard deviations seem to somewhat stabilize at around 100-200 samples for all three cases, where the random female and impersonator 2 appear to be the ones with largest standard deviations. This can also be seen by comparing the widths of their bootstrapped confidence intervals of Table 3 where the bootstrapped means of the accuracies are presented.

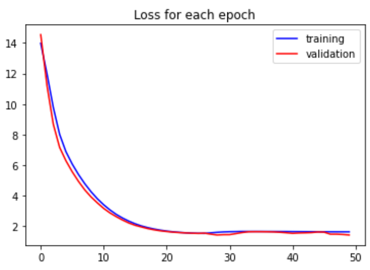
Speakers	kNN $k = 1$		kNN $k = 3$		Random Forest	
	CI	Mean	CI	Mean	CI	Mean
Random male 1	[0.94, 1]	**0.97**	[0.93, 0.99]	0.96	[0.90, 0.98]	0.94
Random male 2	[0.96, 1]	**0.98**	[0.94, 1]	0.97	[0.94, 1]	0.97
Random female	[0.84, 0.92]	**0.88**	[0.58, 0.68]	0.63	[0.56, 0.68]	0.62
Impersonator 1	[0.95, 1]	**0.98**	[0.94, 1]	0.97	[0.94, 1]	0.97
Impersonator 2	[0.55, 0.63]	0.59	[0.58, 0.66]	**0.62**	[0.34, 0.42]	0.38
Impersonator 3	[0.96, 1]	**0.98**	[0.94, 1]	0.97	[0.94, 1]	0.97
Impersonator 4	[0.91, 0.97]	0.94	[0.93, 0.99]	**0.96**	[0.91, 0.97]	0.94
Impersonator 5	[0.95, 1]	**0.98**	[0.95, 1]	0.98	[0.94, 1]	0.97
Impersonator 6	[0.81, 0.87]	0.84	[0.82, 0.88]	**0.85**	[0.75, 0.81]	0.78
True speaker	[0.80, 0.84]	0.82	[0.81, 0.85]	0.83	[0.90, 0.92]	**0.91**
Overall mean	**0.896**		0.874		0.845	

Table 3: Bootstrapped means and confidence intervals of the 100 accuracies (1000 bootstrapped samples), using triplet loss with chosen negative data, of random people speaking as well as impersonators and different recordings of the true speaker (Donald Trump).

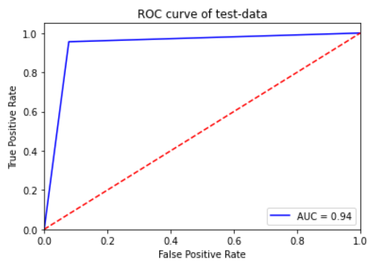
The kNN, $k = 1$ seems to have an overall better performance compared to kNN, $k = 3$ and random forest classifiers, especially when it comes to predicting the random female speaker. Comparing these results to what was obtained with the pair loss, it can be seen that for kNN with $k = 1$ and $k = 3$, the random female's accuracy decreased by 10% and 14%, respectively, whereas for the random forest classifier, it decreased by 36%. The random forest classifier seems to have a hard time classifying the random female and impersonator 2, which can be observed by the significantly lower accuracies for these two speakers, compared to the other two classifiers. However, of the three classifiers, it achieved the highest accuracy when classifying the true speaker.

Quadruplet loss

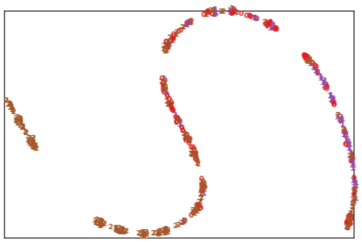
Moving on to when the SNN used the quadruplet loss function, the network's attempt in minimizing the loss function for each epoch, the ROC curve and AUC score, and the tSNE plot of the training data encodings can be seen in Figure 31.



(a) Loss



(b) AUC



(c) tSNE

Figure 31: Training loss (blue) and validation loss (red) after each epoch in (a), ROC curve and AUC score of the 20% test data with AUC = 0.94 in (b), and tSNE plot of the training data encodings of the trained model in (c).

In Figure 31 (a), the loss seems to stabilize at around 20 epochs, and in (b), the ROC curve and AUC score for the 20% test data with $AUC = 0.94$ once again indicate a decent model, and the tSNE plot in (c) shows that the network's attempt to make the distance between the anchor and negative larger while simultaneously making the distance between the anchor and positive smaller. The network does not succeed entirely with the separation, as there are some anchor/negative and positive/negative overlapping, but overall, it seems to indicate a decent model.
The standard deviations of 1000 bootstrapped samples of the means of the accuracies of the speakers in Table 4 and how they converged can be seen in Figure 37 in the Appendix. The standard deviations seem to again stabilize at around 100-200 samples, but seems to overall be smaller, as only impersonator 2 has a standard deviation at around 0.15-0.2, whereas all of the other speakers have 0.1 or less.
This can also be seen by looking at the the width of the bootstrapped confidence intervals in Table 4, where the the bootstrapped means of the accuracies are also presented. Similarly to the network using triplet loss function, the random forest classifier still gave the highest accuracy for the true speaker while falling behind in classifying impersonator 2. However, the SNN with the quadruplet loss seems to be much better at classifying the random female speaker for all three classifiers (98-100% accuracy), and also has an overall better performance.

Speakers	kNN $k = 1$		kNN $k = 3$		Random Forest	
	CI	Mean	CI	Mean	CI	Mean
Random male 1	[0.97, 0.99]	**0.98**	[0.97, 0.99]	**0.98**	[0.93, 0.97]	0.95
Random male 2	[1, 1]	**1**	[1, 1]	**1**	[1, 1]	**1**
Random female	[1, 1]	**1**	[0.97, 0.99]	0.98	[0.97, 0.99]	0.98
Impersonator 1	[1, 1]	**1**	[1, 1]	**1**	[1, 1]	**1**
Impersonator 2	[0.54, 0.60]	0.57	[0.55, 0.63]	**0.59**	[0.34, 0.42]	0.38
Impersonator 3	[1, 1]	**1**	[1, 1]	**1**	[0.99, 1]	**1**
Impersonator 4	[0.94, 0.96]	0.95	[0.96, 0.98]	**0.97**	[0.96, 0.98]	**0.97**
Impersonator 5	[1, 1]	**1**	[1, 1]	**1**	[1, 1]	**1**
Impersonator 6	[0.88, 0.92]	0.90	[0.90, 0.94]	**0.92**	[0.80, 0.84]	0.82
True speaker	[0.78, 0.80]	0.79	[0.77, 0.79]	0.78	[0.90, 0.92]	**0.91**
Overall mean	0.919		**0.922**		0.901	

Table 4: Bootstrapped means and confidence intervals of the 100 accuracies (1000 bootstrapped samples), using quadruplet loss with <u>chosen</u> negative data, of random people speaking as well as impersonators and different recordings of the true speaker (Donald Trump).

Comparison of classifiers and network structures

To compare the performance of the three classifiers, kNN for $k = 1$, kNN for $k = 3$, and random forest, the highest accuracies for each speaker in each of the three network structures are presented in Table 5.

Speakers	Pair loss	Triplet loss	Quadruplet
Random male 1	$\mathbf{0.98}_{***}$	0.97_{kNN1}	$\mathbf{0.98}_{**}$
Random male 2	$\mathbf{1}_{***}$	0.98_{kNN1}	$\mathbf{1}_{***}$
Random female	0.98_{**}	0.88_{kNN1}	$\mathbf{1}_{kNN1}$
Impersonator 1	$\mathbf{1}_{***}$	0.98_{kNN1}	$\mathbf{1}_{***}$
Impersonator 2	0.57_{kNN1}	$\mathbf{0.62}_{kNN3}$	0.59_{kNN3}
Impersonator 3	$\mathbf{1}_{***}$	0.98_{kNN1}	$\mathbf{1}_{***}$
Impersonator 4	$\mathbf{0.98}_{kNN3}$	0.96_{kNN3}	0.97_{**}
Impersonator 5	$\mathbf{1}_{***}$	0.98_{kNN1}	$\mathbf{1}_{***}$
Impersonator 6	0.83_{kNN3}	0.85_{kNN3}	$\mathbf{0.92}_{kNN3}$
True speaker	0.77_{RF}	$\mathbf{0.91}_{RF}$	$\mathbf{0.91}_{RF}$
Best Overall	0.907_{**}	0.896_{kNN1}	$\mathbf{0.922}_{kNN3}$

Table 5: Best accuracies from network structures corresponding to three loss functions. $_{***}/_{**}$ means three/two classifiers got the same best results.

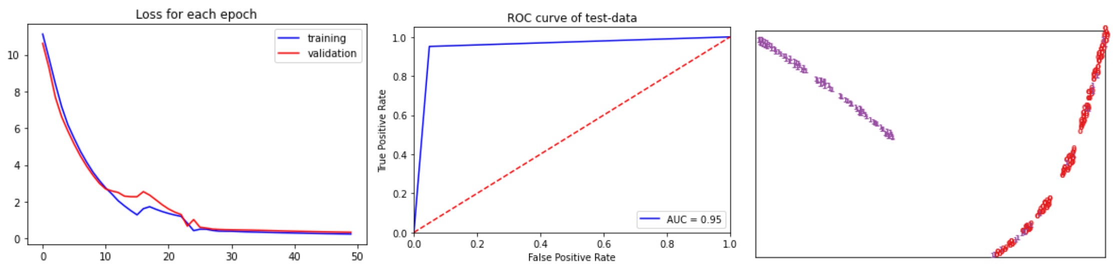
As seen from Table 5, the network structure with quadruplet loss function seems to perform the best, with eight out of ten best scores. Of all the best scores in the network structure using quadruplet loss, classifier kNN, $k = 3$ seems to perform the best with the highest overall mean accuracy as well as highest accuracies for the majority of the test cases. The network structure that has the lowest overall mean accuracy is the one using triplet loss. However, it also has the best accuracy for impersonator 2, at 62%. The network structure using pair loss function has very similar performance to the one using quadruplet loss, the best overall average is only about 2% lower. In conclusion, the network structure using quadruplet loss and classifier kNN, $k = 3$ gives the best results for speaker Donald Trump.

4.2 Speaker Barack Obama

Repeating the procedure as in section 4.1, now with Barack Obama as the true speaker, and only using chosen recordings in the negative data, in this section, the results for SNN using different loss functions are presented.

Pair loss

In Figure 32, the network's attempt in minimizing the loss function for each epoch when using the pair loss function, the ROC curve, AUC score, as well as the tSNE plot of the training data encodings can be observed.



(a) Loss **(b)** AUC **(c)** tSNE

Figure 32: Training loss (blue) and validation loss (red) after each epoch in (a), ROC curve and AUC score of the 20% test data with AUC = 0.95 in (b), and tSNE plot of the training data encodings of the trained model in (c).

The minimization of the loss seems to first run in to some troubles at around epoch 15, but quickly recovers and seems to stabilize at around 30 epochs, as can be seen in Figure 32 (a). The ROC curve and AUC score in (b) also seem to indicate a decent model with $AUC = 0.95$. Moreover, from the tSNE plot in (c) of the training data encodings, it seems as if the network managed to separate the anchor (red zeros) and negative (purple ones) quite well.
The standard deviations of the 1000 bootstrapped samples of the means of the accuracies of the speakers in Table 6 and how they converged can be seen in Figure 38 in the Appendix. The standard deviations seem to need around 300 samples before it starts to stabilize. However, there does not seem to be any clear difference for the three classifiers as they behave quite similarly and have roughly the same

standard deviations.
This can also be seen from the width of the bootstrapped confidence intervals in Table 6, where the bootstrapped means of the accuracies are also presented.

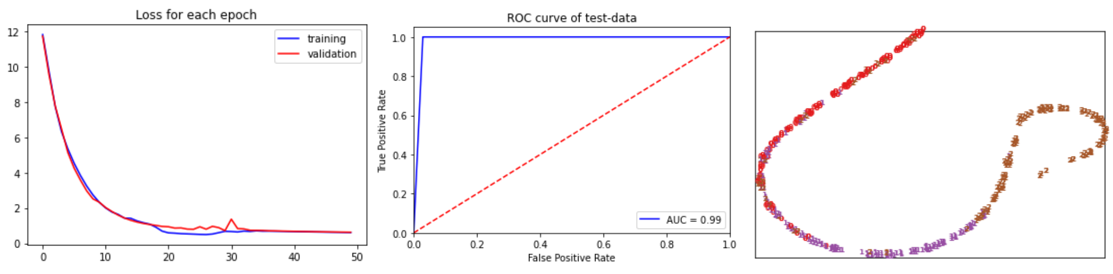
Speakers	kNN $k = 1$		kNN $k = 3$		Random Forest	
	CI	Mean	CI	Mean	CI	Mean
Random male 1	[0.85, 0.97]	0.91	[0.91, 0.99]	**0.95**	[0.85, 0.97]	0.91
Random male 2	[0.77, 0.89]	**0.83**	[0.42, 0.54]	0.48	[0.77, 0.89]	**0.83**
Random female	[0.85, 0.97]	0.91	[0.91, 0.99]	**0.95**	[0.85, 0.97]	0.91
Impersonator 1	[0.86, 0.96]	0.91	[0.88, 0.96]	**0.92**	[0.88, 0.96]	**0.92**
Impersonator 2	[0.87, 0.97]	0.92	[0.88, 0.98]	**0.93**	[0.88, 0.98]	**0.93**
Impersonator 3	[0.85, 0.97]	0.91	[0.89, 0.99]	**0.94**	[0.87, 0.97]	0.92
Impersonator 4	[0.85, 0.97]	0.91	[0.89, 0.99]	**0.94**	[0.87, 0.97]	0.92
Impersonator 5	[0.85, 0.97]	0.91	[0.89, 0.99]	**0.94**	[0.87, 0.97]	0.92
Impersonator 6	[0.87, 0.97]	0.92	[0.88, 0.98]	**0.93**	[0.88, 0.98]	**0.93**
True speaker	[0.50, 0.58]	0.54	[0.51, 0.59]	**0.55**	[0.50, 0.56]	0.53
Overall mean	0.867		0.853		**0.872**	

Table 6: Bootstrapped means and confidence intervals of the 100 accuracies (1000 bootstrapped samples), using quadruplet loss, of random people speaking as well as impersonators and different recordings of the true speaker.

At first the accuracies look rather promising, except for random male 2 for the classifier kNN $k = 3$. The accuracy of the true speaker, i.e., Barack Obama, was also not very promising. It seems as if the network, no matter the classifier, is more happy to predict that the speaker is not Obama, even when the speaker is Obama.

Triplet loss

Figure 33 shows the networks attempt in minimizing the loss function for each epoch when using the triplet loss function, as well as the ROC curve and AUC score, and the tSNE plot of the training data encodings.



(a) Loss **(b)** AUC **(c)** tSNE

Figure 33: Training loss (blue) and validation loss (red) after each epoch in (a), ROC curve and AUC score of the 20% test data with AUC = 0.99 in (b), and tSNE plot of the training data encodings of the trained model in (c).

Just as before, the minimization of the loss in Figure 33 (a) seems to be rather smooth for both the training and the validation data. Other than a small bump at epoch 30, it quickly recovers and stabilizes. The ROC curve and AUC score in (b) with $AUC = 0.99$ also seem to indicate a decent model. Furthermore, the tSNE plot in (c) also shows that the network managed to separate the anchor from the negative to some degree.

The standard deviations of the 1000 bootstrapped samples of the means of the accuracies of the speakers in Table 7 and how they converged can be seen in Figure 39 in the Appendix. The standard deviations for the classifier kNN $k = 1$ seem to be most stable out of the three classifiers, but they all seem to somewhat stabilize at around 300-400 samples. The standard deviations are also smaller for all three classifiers compared to the standard deviations of the pair loss.

This can also be seen from the width of the bootstrapped confidence intervals in Table 7, where the the bootstrapped means of the accuracies are also presented.

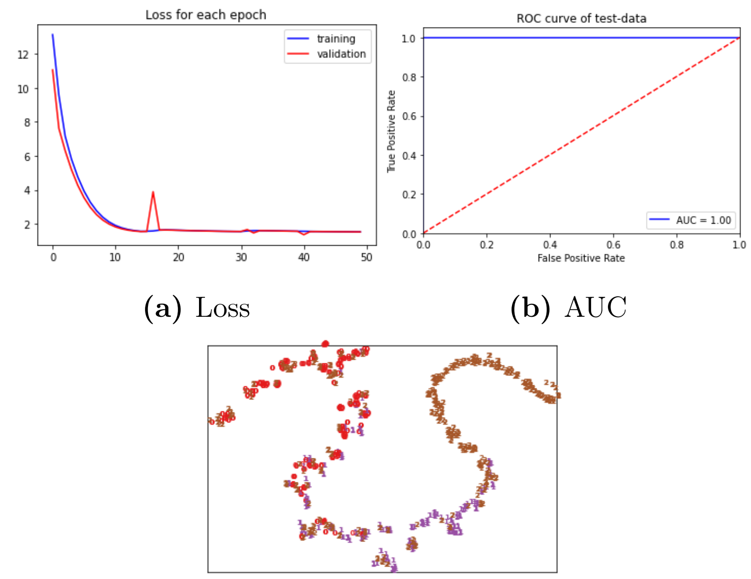
Speakers	kNN $k = 1$		kNN $k = 3$		Random Forest	
	CI	Mean	CI	Mean	CI	Mean
Random male 1	[0.97, 1]	**0.99**	[0.96, 1]	0.98	[0.94, 1]	0.97
Random male 2	[0.98, 1]	**0.99**	[0.88, 0.94]	0.91	[0.88, 0.96]	0.92
Random female	[0.95, 0.99]	0.97	[0.96, 1]	**0.98**	[0.92, 0.98]	0.95
Impersonator 1	[0.69, 0.73]	0.71	[0.70, 0.74]	**0.72**	[0.66, 0.72]	0.69
Impersonator 2	[0.95, 1]	0.98	[0.97, 1]	**0.99**	[0.95, 1]	0.98
Impersonator 3	[0.93, 0.99]	**0.96**	[0.93, 0.99]	**0.96**	[0.90, 0.98]	0.94
Impersonator 4	[1, 1]	**1**	[1, 1]	**1**	[0.95, 1]	0.98
Impersonator 5	[0.99, 1]	**1**	[0.99, 1]	**1**	[0.95, 1]	0.98
Impersonator 6	[0.95, 1]	**0.98**	[0.95, 1]	**0.98**	[0.95, 1]	**0.98**
True speaker	[0.79, 0.83]	0.81	[0.78, 0.82]	0.80	[0.83, 0.85]	**0.84**
Overall mean	**0.939**		0.932		0.923	

Table 7: Bootstrapped means and confidence intervals of the 100 accuracies (1000 bootstrapped samples), using quadruplet loss, of random people speaking as well as impersonators and different recordings of the true speaker.

As seen from Table 7, the accuracies for both random male 2 as well as the true speaker have increased, and the overall results for all three classifiers look rather promising. However, compared with the results from the SNN using pair loss, impersonator 1 has gotten worse by 20%, which seems like decent trade-off considering the 26%, 53% and 9% increase for random male 2 for kNN $k = 1$, $k = 3$, and random forest, respectively. Moreover, there are 27%, 25%, and 31% increases on the true speaker, increasing the overall performance for all three classifiers.

Quadruplet loss

Lastly, the network's attempt in minimizing the loss function for each epoch, the ROC curve and AUC score, and tSNE plot of the training data encodings for the SNN using the quadruplet loss can be seen in Figure 34, respectively.



(a) Loss **(b)** AUC

(c) tSNE

Figure 34: Training loss (blue) and validation loss (red) for each epoch in (a), ROC curve and AUC score of the 20% test data with AUC = 1 in (b), and tSNE plot of the training data encodings of the trained model in (c).

Other than the sudden increase of the loss in Figure 34 (a) at around epoch 18 of the validation data, the minimization seems smooth, and stabilized at around 20 epochs. The ROC curve and AUC score of $AUC = 1$ indicate a very good model. Looking at the tSNE plot of the training data encodings, the network seemed to succeed in separating the anchor (red zeros) and positive (purple ones) from the negative (brown twos). However, there are still quite a lot of overlapping anchor/negatives as well as overlapping positive/negatives.
The standard deviations of the 1000 bootstrapped samples of the means of the accuracies of the speakers in Table 8 and how they converged can be seen in Figure 40 in the Appendix. The main difference of the standard deviations for the three classifiers are observed with random male 2 and impersonator 2. Other than that, the other speakers have standard deviations of around 0.1. Moreover, they all seem to start converging at around 200 samples.
The bootstrapped means and confidence intervals of the accuracies can be seen in Table 8.

Speakers	kNN $k = 1$		kNN $k = 3$		Random Forest	
	CI	Mean	CI	Mean	CI	Mean
Random male 1	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**
Random male 2	[0.97, 1]	**0.99**	[0.88, 0.94]	0.91	[0.92, 0.98]	0.95
Random female	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**
Impersonator 1	[0.74, 0.78]	**0.76**	[0.74, 0.78]	**0.76**	[0.74, 0.78]	**0.76**
Impersonator 2	[0.94, 1]	0.97	[0.95, 1]	**0.98**	[0.95, 1]	**0.98**
Impersonator 3	[0.96, 1]	**0.98**	[0.96, 1]	**0.98**	[0.96, 1]	**0.98**
Impersonator 4	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**
Impersonator 5	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**
Impersonator 6	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**	[0.97, 1]	**0.99**
True speaker	[0.58, 0.60]	0.59	[0.58, 0.60]	0.59	[0.69, 0.71]	**0.7**
Overall mean	0.924		0.917		**0.932**	

Table 8: Bootstrapped means and confidence intervals of the 100 accuracies (1000 bootstrapped samples), using quadruplet loss, of random people speaking as well as impersonators and different recordings of the true speaker.

All three classifiers have very similar result, where they mainly differ on the true speaker. Compared to the SNN using triplet loss function, impersonator 1 got a slight increase (5% with kNN $k = 1$, 4% with kNN $k = 3$, 7% with RF); however, at the cost of the true speaker's decrease by 14% for the random forest and 21-22% for the kNN classifiers. This also leads to the slightly worse overall mean.

Comparison of classifiers and network structures

To compare the performance of the three classifiers, kNN for $k = 1$, kNN for $k = 3$ and random forest, the highest accuracies for each speaker in each of the three network structures are taken out and presented in Table 9.

Speakers	Pair loss	Triplet loss	Quadruplet
Random male 1	0.95_{kNN3}	$\mathbf{0.99}_{kNN1}$	$\mathbf{0.99}_{**}$
Random male 2	0.83_{**}	$\mathbf{0.99}_{kNN1}$	$\mathbf{0.99}_{kNN1}$
Random female	0.95_{kNN3}	0.98_{kNN3}	$\mathbf{0.99}_{***}$
Impersonator 1	$\mathbf{0.92}_{**}$	0.72_{kNN3}	0.76_{***}
Impersonator 2	0.93_{**}	$\mathbf{0.99}_{kNN3}$	0.98_{**}
Impersonator 3	0.94_{kNN3}	0.96_{kNN3}	$\mathbf{0.98}_{***}$
Impersonator 4	0.98_{kNN3}	$\mathbf{1}_{**}$	0.99_{***}
Impersonator 5	0.94_{kNN3}	$\mathbf{1}_{**}$	0.99_{***}
Impersonator 6	0.93_{**}	0.98_{**}	$\mathbf{0.99}_{***}$
True speaker	0.55_{kNN3}	$\mathbf{0.84}_{RF}$	0.70_{RF}
Best Overall	0.872_{RF}	$\mathbf{0.939}_{kNN1}$	0.932_{RF}

Table 9: Best accuracies from network structures corresponding to three loss functions. $_{***}/_{**}$ means three/two classifiers got the same best results.

By looking at Table 9, the performances of different network structures can also be compared. Only comparing the best overall mean accuracy, the network structure using triplet loss function seems to be the best. However, the one with quadruplet loss is really not far behind, the best overall is lower by only less than 1%, and for impersonator 1, the accuracy is better by 4%. Meanwhile, for the majority of all the tests, the network with triplet loss is always slightly better, and for the true speaker, it is significantly better than the other network structures with 14% higher mean accuracy than that of the network structure with quadruplet loss. The network structure that has the lowest overall score is the one with pair loss function. For nine out of ten test speakers, it performs slightly worse or much worse (the true speaker test) than the other two network structures. Nevertheless, it surprisingly did really well on impersonator 1, with an accuracy of 92%, 20% higher than the network with triplet loss. For the performance of the classifiers, for network structures with pair loss and quadruplet loss, random forest classifier seem to perform the best in overall. For the network structure with triplet loss, kNN, $k = 1$ seems to perform the best, with kNN, $k = 3$ not so far behind. In conclusion, it can be said that the network structure with triplet loss function, and classifier kNN, $k = 1$ gives us the best results for speaker Barack Obama.

5 Conclusion

The goal of this thesis was to classify a person's voice and their impersonators. In order to achieve that, audio features from the voice recordings were extracted and fed into a Siamese Neural Network to generate encodings. These encodings were then used as inputs for the random forest and kNN classifiers to predict whether the corresponding recordings belonged to the true speaker or not.

For each recording, we studied different audio features extracted from the five-second segments as well as the 20- and 40-millisecond segments containing periodic structures, and the different combinations of these features. The results showed that concatenating features from the harmonic 20- and 40-millisecond segments with those from the five-second segments into a single feature matrix was preferable. For the millisecond segments, the MFCCs, the first four formants, and the formant ratios between the second and first formants were extracted. At the same time, for the five-second segments, the spectrograms with higher resolution in frequency were computed. Moreover, of the network architectures that were tested, the one that performed the best consisted of a two-dimensional convolution layer with a non-square kernel, a two-dimensional max pooling layer, a two-dimensional global average pooling layer, three dense layers, and a batch normalization layer.

The model was tested on two speakers, Donald Trump and Barack Obama, and evaluated by computing the accuracy, i.e., how many samples were correctly classified out of all the samples for each speaker, including both the true speakers and their impersonators. In the first case, with Donald Trump as the true speaker, the loss function and classifier that resulted in giving the overall highest accuracies was the quadruplet loss function with classifier kNN with $k = 3$ neighbors. In the second case, with Barack Obama as the true speaker, the loss function and classifier that resulted in giving the overall highest accuracies was the triplet loss function with classifier kNN with $k = 1$ neighbors. The different results for the two different true speakers could be explained by different reasons. One reason could be that the recordings that were tested and picked out for the chosen negative data set were better suited for either of the two cases. Another reason could be due to the fact that within the scope of this thesis, more data of Trump's impersonators were gathered, as compared to Obama. This makes it more likely that some of Trump's impersonators were better and thus, harder to classify.

For both Trump and Obama, the classifiers that gave the highest accuracies, no matter the loss function, did not seem to have any major difficulties classifying the random males and female correctly. Furthermore, some impersonators were easier to classify than the others. We suspect that the dissimilarities between these people and the true speaker already resulted in the different values in the spectrograms, MFCCs, and other features in the raw feature matrix. For the impersonators that sounded very similar to the true speakers, they also had similar feature values, which made it harder for the network to separate them.

6 Further study

For further study, to evaluate the model's performance, it would be a good idea to collect more data from other speakers and their impersonators. There are other prominent political figures in the U.S. that also have been impersonated for comedic purposes, such as Hilary Clinton, Joe Biden, and George W. Bush. Additionally, it would be interesting to test the model on data from people speaking other languages than American English and their impersonators. Another data set of interest could be recordings from twins where the classification task then is to differentiate between them.

Furthermore, what could improve the performance lies in the extraction of periodic structures from an audio recording. Our current method discards a number of segments that have perfectly periodic structures, which could result in us missing important unique features for each speaker. A better method to ensure all periodic structures are kept and used in the training could affect the results positively. Moreover, if specific vowels for each speaker could be extracted and tagged, one could build the raw feature matrix in a different way, for example, each column can be features extracted from one specific vowel. Our thought is that it might be easier for the network to distinguish between speakers when they say the same thing.

Finally, as choosing a negative data set can impact the performance of the network quite significantly, it might also be a good idea to investigate other functions one could use in measuring the distances between voices, and also different types of inputs for the distance function. For example, if specific vowels can be extracted as explained above, one could immediately compute the distance from the raw feature matrix, instead of computing the distance between the encodings as in our current method. Moreover, for computing the distance between speakers saying the same vowels, there are other distance functions one can look into.

References

[1] N. Almaadeed, A. Aggoun, and A. Amira. “Text-Independent Speaker Identification Using Vowel Formants.” In: *J Signal Processing System 82* (2016), pp. 345–356. DOI: 10.1007/s11265-015-1005-5.

[2] N.S. Altman. “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression”. In: *The American Statistician* 46.3 (1992), pp. 175–185. DOI: 10.1080/00031305.1992.10475879. eprint: https://www.tandfonline.com/doi/pdf/10.1080/00031305.1992.10475879. URL: https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879.

[3] T. Bäckström et al. *Introduction to Speech Processing.* Aalto University Wiki, 2022. URL: https://wiki.aalto.fi/display/ITSP/Introduction+to+Speech+Processing.

[4] P. Bloomfield. *Fourier Analysis of Time Series: An Introduction.* New York: Wiley-Interscience, 2000. ISBN: 0-471-88948-2.

[5] P. Boersma. “Accurate Short-term Analysis of the Fundamental Frequency and the Harmonics-to-noise Ratio of a Sample Sound.” In: *Proceedings 17* (1993), pp. 97–110. URL: https://www.fon.hum.uva.nl/david/ba_shs/2010/Boersma_Proceedings_1993.pdf.

[6] P. Boersma and D. Weenink. *Praat: doing phonetics by computer (Version 5.1.13).* 2009. URL: http://www.praat.org.

[7] F. Bre, J. M. Gimenez, and V. D. Fachinotti. “Prediction of wind pressure coefficients on building surfaces using artificial neural networks.” In: *Energy and Buildings 158* (2018), pp. 1429–1441. URL: https://doi.org/10.1016/j.enbuild.2017.11.045.

[8] A. Cephei. *Vosk Speech Recognition Toolkit.* Version 0.3.31. URL: https://alphacephei.com/vosk/.

[9] W. Chen et al. “Beyond triplet loss: a deep quadruplet network for person re-identification”. In: *CoRR* abs/1704.01719 (2017). URL: http://arxiv.org/abs/1704.01719.

[10] “Coefficient of Determination”. In: *The Concise Encyclopedia of Statistics.* New York, NY: Springer New York, 2008, pp. 88–91. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_62. URL: https://doi.org/10.1007/978-0-387-32833-1_62.

[11] J. Fernandes et al. "Harmonic to Noise Ratio Measurement - Selection of Window and Length." In: *Procedia Technology 138* (2018), pp. 280–285. URL: https://doi.org/10.1016/j.procs.2018.10.040.

[12] M. Hansson. "Optimized Weighted Averaging of Peak Matched Multiple Window Spectrum Estimates". In: *IEEE Transactions on Signal Processing* 47.4 (1999), pp. 1141–1146. ISSN: 1941-0476. DOI: 10.1109/78.752613. URL: https://ieeexplore.ieee.org/document/752613.

[13] Y. Jadoul, B. Thompson, and de Boer B. "Introducing Parselmouth: A Python interface to Praat". In: *Journal of Phonetics* 71 (2018), pp. 1–15. DOI: https://doi.org/10.1016/j.wocn.2018.07.001.

[14] A. Jakobsson. *An Introduction to Time Series Modeling.* Studentlitteratur, 2013, pp. 53–54. ISBN: 978-91-44-13403-1. URL: https://www.studentlitteratur.se/kurslitteratur/matematik-och-statistik/matematisk-statistik/an-introduction-to-time-series-modeling.

[15] D.P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization.* 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[16] G. Koch, R. Zemel, and R. Salakhutdinov. "Siamese Neural Networks for One-shot Image Recognition". In: (2015). URL: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

[17] J. Luo, J. Zhang W. an Su, and F. Xiang. "Hexagonal Convolutional Neural Networks for Hexagonal Grids". In: *IEEE Access* PP (Sept. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2944766. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8853238.

[18] L. van der Maaten and G. Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[19] MATLAB. *Formant Estimation with LPC Coefficients.* Version R2022a. URL: https://se.mathworks.com/help/signal/ug/formant-estimation-with-lpc-coefficients.html.

[20] P. Munro. "Backpropagation". In: *Encyclopedia of Machine Learning.* Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 73–73. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_51. URL: https://doi.org/10.1007/978-0-387-30164-8_51.

[21] S. Murray et al. “Donald Trump on recording: Not me”. In: (2016). URL: https://edition.cnn.com/2016/05/13/politics/donald-trump-recording-john-miller-barron-fake-press/.

[22] T. O’Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[23] K. Rao and K. E. Manjunath. *Speech Recognition Using Articulatory and Excitation Source Features.* Jan. 2017. ISBN: 978-3-319-49219-3. DOI: 10.1007/978-3-319-49220-9.

[24] M. Sandsten. *Time-Frequency Analysis of Time-Varying Signals and Non-Stationary Processes: An Introduction.* 2022.

[25] J. Schmidhuber. “Deep learning in neural networks: An overview.” In: *Neural Networks 61* (2015), pp. 85–117. URL: https://doi.org/10.1016/j.neunet.2014.09.003.

[26] U. Shrawankar and V. M. Thakare. “Techniques for Feature Extraction In Speech Recognition System : A Comparative Study”. In: (2013). DOI: 10.48550/ARXIV.1305.1145. URL: https://arxiv.org/abs/1305.1145.

[27] V. Stenvall. “Detection of Relapsing Vocal Cord Cancer Using Siamese Neural Networks.” In: (2021). URL: http://lup.lub.lu.se/student-papers/record/9045289.

[28] J.P. Teixeira and A. Goncalves. “Accuracy of Jitter and Shimmer Measurements.” In: *Procedia Technology 16* (2014), pp. 1190–1199. URL: https://doi.org/10.1016/j.protcy.2014.10.134.

[29] D.J. Thomson. “Spectrum estimation and harmonic analysis”. In: *IEEE* 70.9 (1982), pp. 1055–1096. ISSN: 1558-2256. DOI: 10.1109/PROC.1982.12433. URL: https://ieeexplore.ieee.org/document/1456701.

[30] Y.C. Tsao, G. Weismer, and K. Iqbal. “Interspeaker variation in habitual speaking rate: Additional evidence.” In: *Journal of Speech, Language, and Hearing Research* (2006), pp. 1156–1164. DOI: 10.1044/1092-4388(2006/083). URL: https://pubs.asha.org/doi/10.1044/1092-4388%5C%282006/083%5C%29.

Appendix

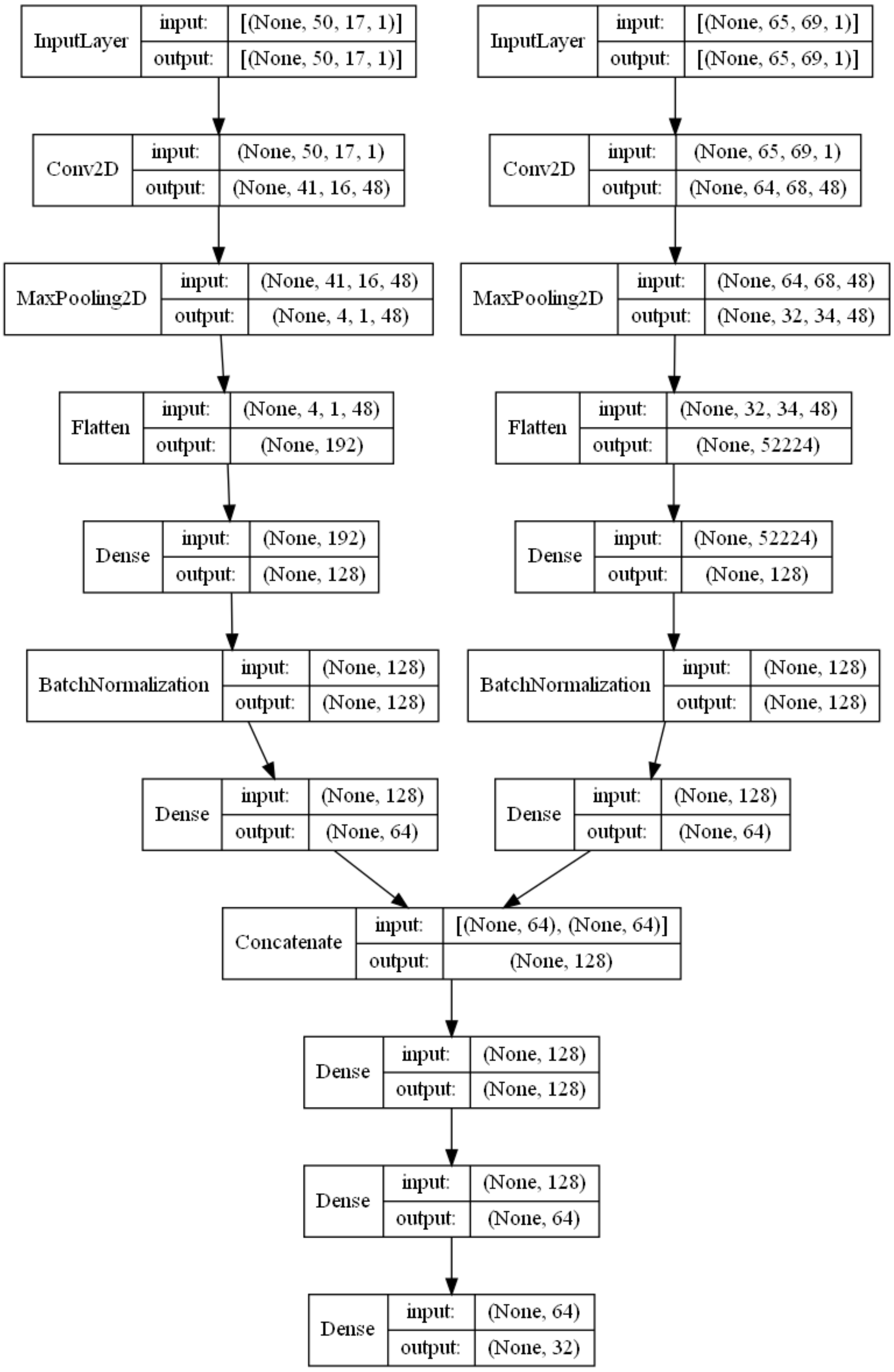
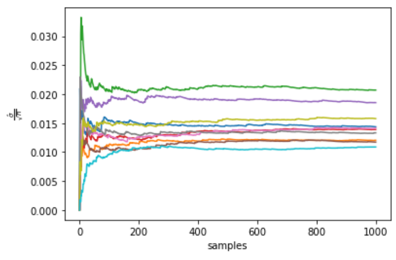


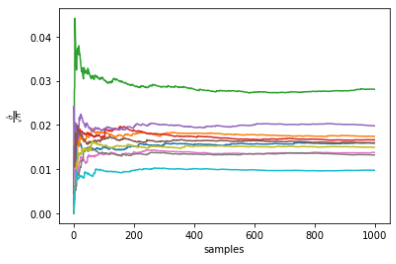
Figure 35: Example of two input sister network architecture, left branch takes the 20ms features, right branch takes 5s features

Speaker	Target	Name
Impersonator 1	Trump	Anthony Atamanuik
Impersonator 2	Trump	Austin Nasso
Impersonator 3	Trump	Bob DiBuono
Impersonator 4	Trump	J-L Cauvin
Impersonator 5	Trump	John Di Domenico
Impersonator 6	Trump	Thomas Mundy
Impersonator 1	Obama	Jordan Peele
Impersonator 2	Obama	Jordan Peele
Impersonator 3	Obama	Jordan Peele
Impersonator 4	Obama	Trevor Noah
Impersonator 5	Obama	Reggie Brown
Impersonator 6	Obama	Steve Bridge

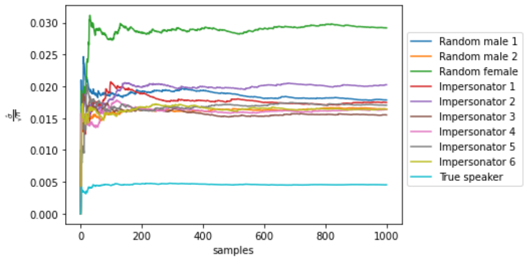
Table 10: True names of the impersonators of Trump and Obama.



(a) kNN $k = 1$

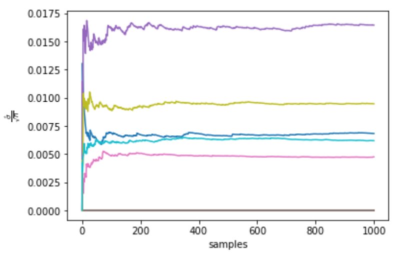


(b) kNN $k = 3$

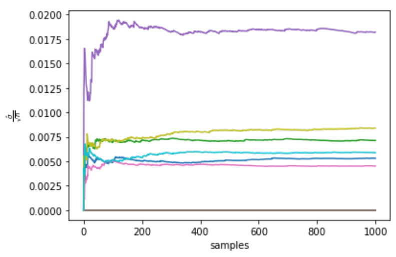


(c) Random forest

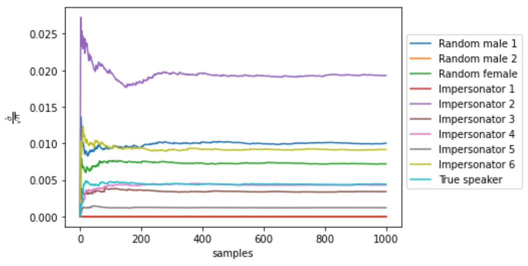
Figure 36: Standard deviations of 1000 bootstrapped (with replacement) samples of the means of the accuracies, using kNN $k = 1$ in (a), kNN $k = 3$ in (b), and random forest in (c). Triplet loss for Donald Trump as true speaker.



(a) kNN $k = 1$

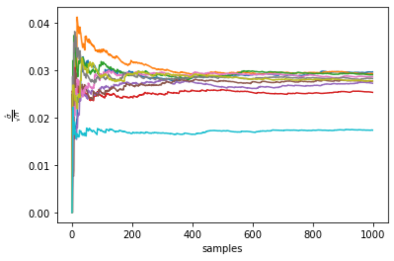


(b) kNN $k = 3$

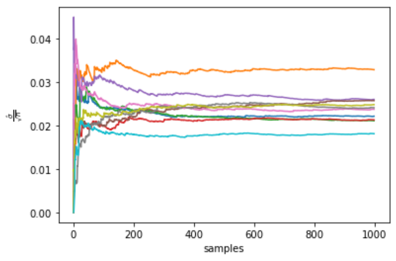


(c) Random forest

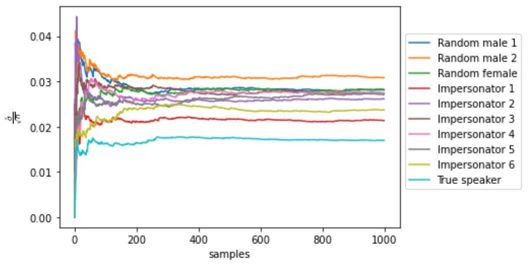
Figure 37: Standard deviations of 1000 bootstrapped (with replacement) samples of the means of the accuracies, using kNN $k = 1$ in (a), kNN $k = 3$ in (b), and random forest in (c). Quadruplet loss for Donald Trump as true speaker.



(a) kNN $k = 1$

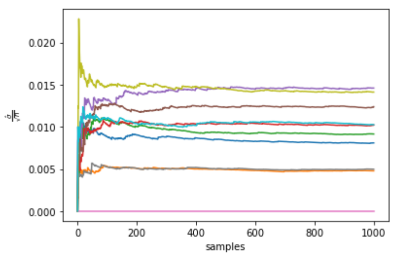


(b) kNN $k = 3$

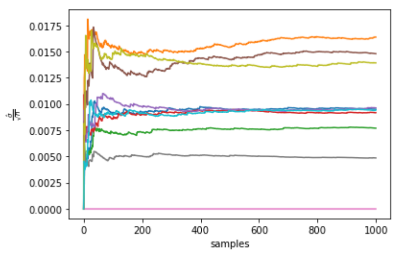


(c) Random forest

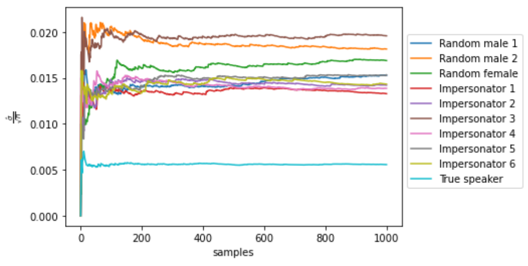
Figure 38: Standard deviations of 1000 bootstrapped (with replacement) samples of the means of the accuracies, using kNN $k = 1$ in (a), kNN $k = 3$ in (b), and random forest in (c). Pair loss for Barack Obama as true speaker.



(a) kNN $k = 1$

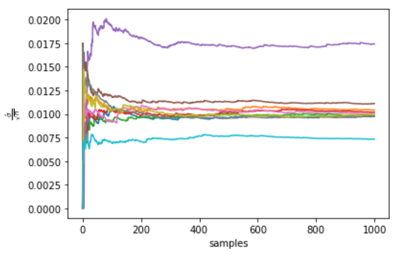


(b) kNN $k = 3$

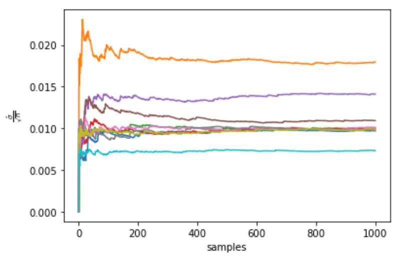


(c) Random forest

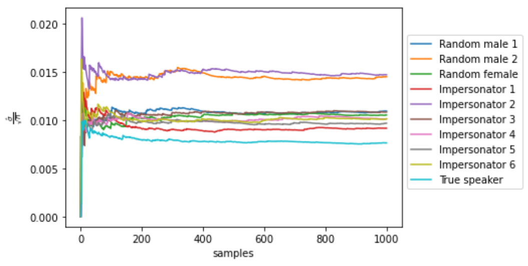
Figure 39: Standard deviations of 1000 bootstrapped (with replacement) samples of the means of the accuracies, using kNN $k = 1$ in (a), kNN $k = 3$ in (b), and random forest in (c). Triplet loss for Barack Obama as true speaker.



(a) kNN $k = 1$



(b) kNN $k = 3$



(c) Random forest

Figure 40: Standard deviations of 1000 bootstrapped (with replacement) samples of the means of the accuracies, using kNN $k = 1$ in (a), kNN $k = 3$ in (b), and random forest in (c). Quadruplet loss for Barack Obama as true speaker.

Master’s Theses in Mathematical Sciences 2022:E35
ISSN 1404-6342

LUNFMS-3109-2022

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/