Detection of relapsing vocal cord cancer using Siamese neural networks

Viktor Stenvall

Master's thesis
2021:E18

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Abstract

In this thesis, we investigate the possibility of using Siamese Neural Networks to detect voice changes in the voices of patients suffering from a recurrence of their vocal cord cancer. In collaboration with VoiceDiagnostic Sweden AB and physicians at Lunds's University Hospital, models were trained on audio features in order to learn distance measures between recordings, and segments of recordings. The resulting models were able to distinguish whether pairs of recordings came from the same or differing users, with an accuracy of over 90%. The best performing frame level model was the Siamese Neural Network using a contrastive loss function. The best recording level model was the network with a binary cross entropy loss function utilising a bi-branch input structure. These models obtained AUC scores of 0.950 and 0.979 respectively, on the validation set.

The models were tested on recordings made by four separate users believed to have experienced voice changes during the recording period. Two of the users had had their vocal cord cancer relapse and the others were experiencing gender dysphoria and in the process of altering their voices together with a speech therapist. There was an observable change in the voices of all four patients according to the frame level model.

Sammanfattning

I denna avhandling undersöker vi möjligheten att använda Siamesiska Neurala Nätverk för att detektera röstförändringar bland patienter som lider av återkommande stämbandscancer. I samarbete med VoiceDiagnostic Sweden AB och läkare vid Lunds Universitetssjukhus konstruerades modeller för att mäta avstånd mellan olika röstinspelningar, samt avstånd mellan midre segment av röstinspelningar. De resulterande modellerna kunde skilja på om inspelningspar härstammade från samma eller olika användare, med en noggrannhet på över 90%. Den bäst presterande modellen på ramnivå var nätverket med en kontrastförlustfunktion (contrastive loss). Den bästa modellen på inspelningsnivå var nätverket med en binär korsentropiförlustfunktion (binary cross entropy loss) som använde en grenad struktur. Dessa modeller uppnådde AUC-resultat på 0,950 respektive 0,979 på valideringsdatan.

Modellerna testades på inspelningar gjorda av fyra separata användare. Dessa fyra användare tros ha upplevt röständringar under inspelningsperioden. Två av användarna hade fått återfall av stämbandscancer, och de andra två upplevde könsdysfori och var i färd med att ändra sina röster tillsammans med logoped. Det skedde en observerbar förändring i rösterna hos alla fyra patienterna enligt den bästa ramnivåmodellen.

Acknowledgements

I would like to extend a big thank you to Johan Swärd and Andreas Jakobsson for all the help you have provided me with during this thesis. Thank you to Andreas for helping me find the vision and overall direction for the thesis. Thank you Johan for the invaluable help surrounding the technical specifics regarding machine learning and Siamese neural networks. And thank you both for always being there as a sounding board for new ideas.

Glossary

ANN	Artificial Neural Network
SNN	Siamese Neural Network
BCE	Binary Cross Entropy
CCE	Categorical Cross Entropy
WBCE	Weighted Binary Cross Entropy
SGD	Stochastic Gradient Descent
MFCC	Mel Frequency Cepstral Coefficients
DFT	Discrete Fourier Transform
DIT	Discrimination In Time
FFT	Fast Fourier Transform
STFT	Short Time Fourier Transform
t-SNE	T-distributed Stochastic Neighbor Embeddings
KL	Kullback-Leibler
TN	True Negatives
TP	True Positives
FN	False Negatives
FP	False Positives
ROC	Receiver Operating Characteristic
AUC	Area Under the ROC Curve

Contents

1 Introduction

Our voices are a key part of our identities and they can disclose information about our health status and well being. Many diseases can for example cause significant changes in a patient's voice over time. An example of such a disease is vocal cord cancer. Cancer of the vocal cords are a subset of Laryngeal cancers, which constitutes about 1% of all yearly diagnosed cancers worldwide [1]. There are three different areas in the larynx where laryngeal cancers can present themselves; the supraglottis, glottis (vocal cords), and subglottis. A visualisation of the larynx and these three areas can be seen in Figure 1.1. Vocal cord (glottis) cancer is a relatively common subset of laryngeal cancers with around 60% of all cases falling within this category [2]. One of the primary symptoms of vocal cord cancer is hoarseness and changes to the voice of the patient. These symptoms, although uncomfortable for the patient, are favorable from a diagnostic standpoint since it often results in earlier detection of the disease, which in turn leads to a larger number of successful treatments. However, these voice changes may also be gradual and incremental in nature, making them difficult to detect for physicians. Physicians have other diagnostic methods at their disposal such as biopsies and endoscopies (laryngoscopies). However, these procedures tend to be rather invasive and uncomfortable for the patient, and not all physicians have access to the tools required to perform them. Furthermore, it may be difficult for the physician to asses when these methods are needed. Physicians could therefore benefit from more detailed data regarding the state of the patient's voice in order to make more informed decisions.

There exists a need for patients and physicians to be able to more closely monitor, detect, and track changes in patients' voices over time. This applies to patients with vocal cord cancer as well as other forms of ailments altering the state of the voice. If we could measure the distances between voice states at different points in time, we could potentially track changes in the voice and detect anomalies regarding those changes. The voice of the sick individual would over time get further away relative to the "healthy" state of the voice, assuming that the progression of the disease worsens the state of the voice. Figures 1.2 and 1.3 display examples of what such scenarios could look like, both if the patient's voice changed over time due to illness (Figure 1.2), and if the voice did not change (Figure 1.3). Providing this sort of granular and sequential data to physicians would enable them to track the progression of the changes and detect relapses of cancer. This technology therefore has the potential to automate patient monitoring and reduce the number of required visits with physicians. This could decrease costs for all parties while providing more accessible and better data enabling better care. This method is currently most viable in cases where the patient has been diagnosed with vocal cord cancer (or an other form of cancer) and is in remission or has recovered fully.

Recent progress made within the field of machine learning has opened up many new possibilities across an array of industries. One such field is medicine. Through vari-

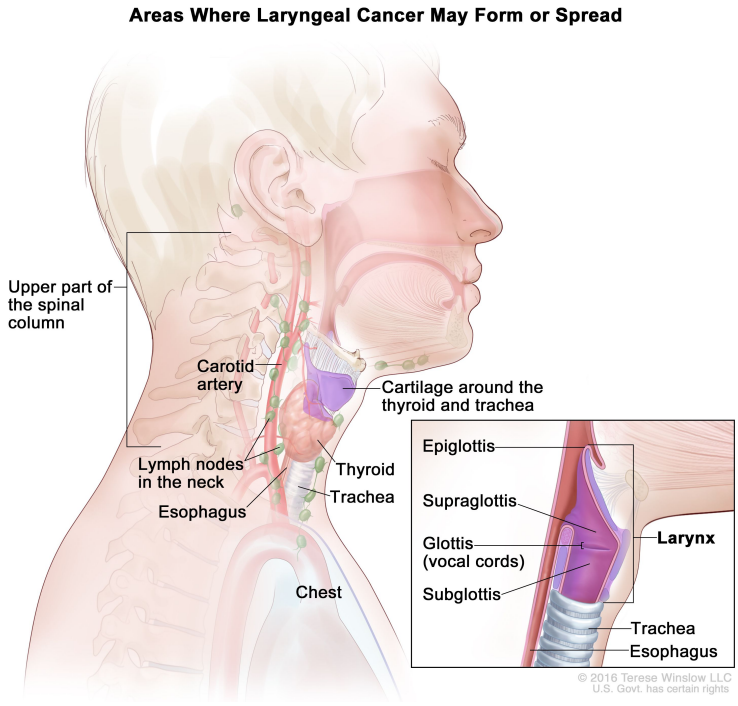

Figure 1.1: Depiction of the larynx in the throat [3]. Laryngeal cancer may form in the supraglottis, glottis, and subglottis. Vocal cord cancer is cancer of the glottis.

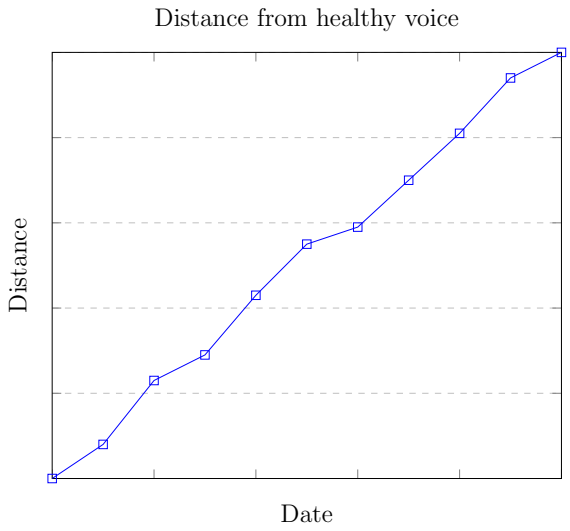

Figure 1.2: Hypothetical scenario depicting the progression of a sick voice.

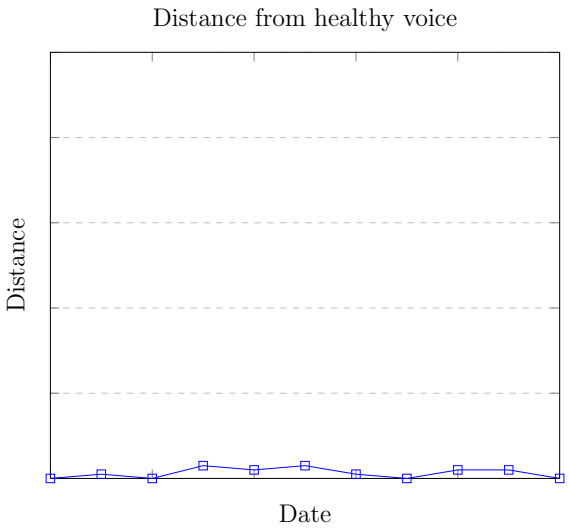

Figure 1.3: Hypothetical scenario depicting the progression of a healthy voice.

ous methods of statistical modelling we now have the tools required to drastically enhance and improve the diagnosis and treatment of various diseases. For example, a research team at the Korean Institute of Science and Technology recently developed a non-invasive technique for diagnosing prostate cancer using machine learning with a recorded accuracy of 99.5% [4]. Machine learning can enable physicians to make more informed and accurate decisions regarding patient health and well-being, potentially saving more lives.

The main limiting factor of machine learning has always been the large amounts of data required to build and scale these models to high performance. Models are often trained to perform one specific task, and even though we have seen stellar performances in a range of tasks over the past few decades, these models do not perform well on tasks which lie outside of their narrow scopes. Extensive research has therefore been conducted into building models which are able to generalise better, on less data. One such field of research is few-shot learning where the focus lies on modeling in cases of limited training data availability. Such models should in theory not require large amounts of data to be able to achieve high accuracies on new data instances. Furthermore, these models do not need to be re-trained when they encounter novel classes and examples they have never before seen.

A breakthrough within few-shot learning was had when Koch et al published their paper on Siamese Neural Networks for One-Shot Image Recognition [5]. Siamese Neural Networks (SNN) are a form of artificial neural network which aim to solve the problem presented by limited data through learning to create an optimal embedding space for inputs. The optimisation of the embedding space aims to bring alike inputs closer to each other in vector space, whilst separating differing inputs from each other. The embeddings produced by the network can in turn be used to distinguish between differing classes of inputs, and to cluster alike data samples. This approach differs from traditional machine learning models which, most often, aim to learn specific features of every class of objects in the dataset in order to be able to distinguish between them. Siamese Networks are now widely used within areas such as facial recognition, signature comparison, and document query matching.

Siamese Neural Networks have in recent years risen to prominence primarily within image classification tasks. However, the broad scope of these models enable their use for a wide range of applications. These types of networks can, and have also, been applied successfully to audio classification tasks [6] [7]. Using Siamese Neural Network with audio data as inputs will be the focus of this thesis.

1.1 Problem Formulation

Throughout this thesis, we aim to investigate if Siamese Neural Networks can be used in order detect voice changes in patient suffering from recurring vocal cord cancer. In order to reach this goal, we aim to answer the following questions:

1. Can Siamese Neural Networks be used to track changes in a person's voice over time?

2. What network architectures are most effective in distinguishing between voices and detecting variations in voices over time?

3. What features of a voice are most important when trying to track changes to that voice over time?

1.2 Previous Work

Since laryngeal cancers account for a low percentage of all cancers worldwide, and patients have relatively high survival rates, it is not the most researched cancer form with regard to diagnostic tools and treatment of the disease. However, it is clear than the voice changes often experienced during the progression of the disease makes it easier to detect. There has been certain research conducted into the field of automatic disease detection of the cancer. However, the progress has been hindered due to the lack of publicly available datasets. The studies which have been conducted are often performed on a smaller sample of patients diagnosed with the disease which can lead to lower performance outcomes and less reliable models.

Due to recent progress made within the field of image classification, many automatic diagnostic models utilise images and scans in the form of MRIs, CTs, X-rays, and ultrasounds, in order to make their diagnosis. This is the case for most studies related to laryngeal cancer detection. For example, in a study from 2019 by Xiong et al [8], deep convolutional neural networks were used to detect tumors and lesions in the larynx based on images from laryngoscopies. A laryngoscopy is an exam of the larynx where a camera is inserted into the throat of the patient in order to take images and tissue samples for further investigative and diagnostic purposes. The trained model could in this case detect, with an overall accuracy of 86.7%, laryngeal tumors, and if those tumors were benign or malignant, based only on images from a patient's laryngoscopy.

Certain research has also been conducted into the field of vocal cord cancer diagnosis using audio features as opposed to scans and other medical imaging. For example, Kim et al [9] used convolutional neural networks applied to audio recording features in order to detect laryngeal cancer with an accuracy of 85%.

As far as the author of this thesis is aware, Siamese Neural Networks have not yet been applied to the field of vocal cord cancer detection. This makes it a novel approach to the problem. However, the lack of research within this field regrettably makes it difficult to compare the results obtained in this thesis to those of other publications.

1.3 Contribution From This Work

Most of the previous research conducted into the diagnosis of various voice pathologies does not make use of audio recordings in order to make the diagnoses. This is in part due to limited data availability, but also due to medical imaging and biopsies being the standard diagnostic tools used today. Furthermore, there are no studies in which SNNs have been utilised to diagnose vocal cord cancer. SNNs applied to audio features therefore presents a novel approach to the problem and a potentially useful tool which can be used by physicians to help the diagnosis of recurring vocal cord cancer. The models may easily be extended to other forms of diseases which cause voice changes, making this approach very versatile.

1.4 Technologies

It has never been easier to get started with machine learning. There are many packages and libraries available today in most programming languages, which enable efficient and scalable computing tools. There is a plethora of open source material related to machine learning and a thriving community built around open source. Furthermore, there are many different dataset publicly available which can be used to train, validate, and test ones machine learning models. The following tools and technologies were used in this project in order to pre-process the data, extract audio features, and build and test the models.

1.4.1 MATLAB

MATLAB is a high-performance programming language for technical computing. The language is typically used for computationally intensive mathematical operations and algorithmic development. There are several toolboxes available in MATLAB which enable the processing of audio data. In particular, the Audio toolbox is used throughout this project in order to pre-process and extract features of audio files that are to be used to train and evaluate the machine learning models.

1.4.2 Python

Python is an interpreted, high-level and general-purpose programming language. Due to its simplicity and availability, Python has cemented itself as the go-to language for data science and machine learning. There are numerous libraries available in Python for the processing of large datasets and packages which are highly efficient at matrix computation; precisely the type of mathematical operations required when building and training machine learning models.

Packages such as Pandas make it easy to load, pre-process, and post-process data efficiently. Numpy, another open source package for handling array computation in python, is a vital tool for data science. The package SciKit-Learn includes a selection of algorithms and tools which can be used to evaluate the models. An example of such an algorithm which is used in this thesis is the t-distributed stochastic neighbor embedding (t-SNE). Furthermore, Librosa is a python package for music and audio analysis which can be useful when loading and processing audio files.

1.4.3 Tensorflow/Keras

Tensorflow is a free and open source machine learning library developed by Google Brain which enables the modelling and training of deep neural networks. It is available on most operating systems today. The library is very flexible in the types of architectures one can build and how they are to be trained. This enables the building of relatively complex and highly customisable models. Tensorflow can run on both CPUs and GPUs with the potential to speed up training times.

Keras is a high level deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation and ease of use.

1.5 Report Outline

This report begins with a chapter on the theoretical background, followed by the method, the results and discussion regarding those result, and last of all, conclusions and suggestions for further work. The theoretical background focuses on providing a detailed description and the theory behind the relevant machine learning and audio processing concepts used in the thesis. The chapter named method breaks down the processes of data gathering, data pre-processing and feature extraction, network modelling, and evaluation of the machine learning models. In the results and discussion chapter, the results of the research and the performance of the models is presented and assessed through discussion. Finally, the conclusion discusses the overall success of the project with relation to the underlying scope of the thesis and the problem formulation. Furthermore, potential future work which could be performed is discussed at the end of the thesis.

2 Theoretical Background

This chapter serves as an in depth review of the essential technologies used, their purpose, and the theoretical background as to how they function.

2.1 Audio Feature Extraction

In order to understand the process of audio feature extraction from digital audio recordings, we first introduce some background information on how these recordings are made.

An audio recording is a representation of the waves which make up what we interpret as sound. Today, most audio recordings are made digitally with recording devices such as smartphones. Digital recording devices use microphones and other hardware combined with specific software in order to create a digital representation of the audio signal. This occurs through the process of sampling and quantisation where the device records, at specific time intervals, the amplitude of the sound waves and stores it digitally. This makes the audio signal representation discrete and finite, and easier to manipulate. The process of quantisation (discretisation) and sampling can be seen in Figure 2.1. Sound waves are typically sampled at a rate of 8, 44, or 48 kHz depending on the application and use case. A lower sampling rate leads to less data required to represent the signal and therefore less space required to store the data. A lower sampling rate also leads to lower audio quality. The same holds true for the number of quantisation levels present in the representation.

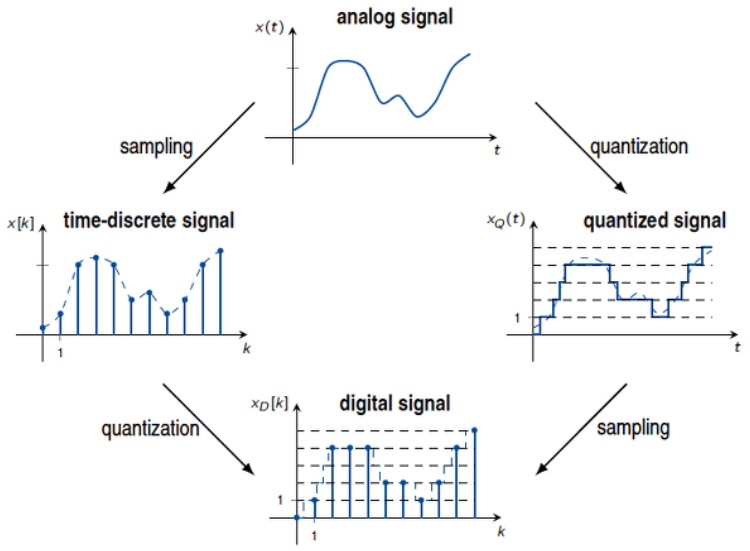

Figure 2.1: The process of sampling and quantising an audio signal. [10]

The features of audio time series signals are often represented in two distinct domains;

the time domain and the frequency domain. Both domains examine the amplitude of the signal, however the former does so with regard to time and the latter with regard to the various frequencies present in the signal. A representation of these domains can be seen depicted in Figure 2.2.

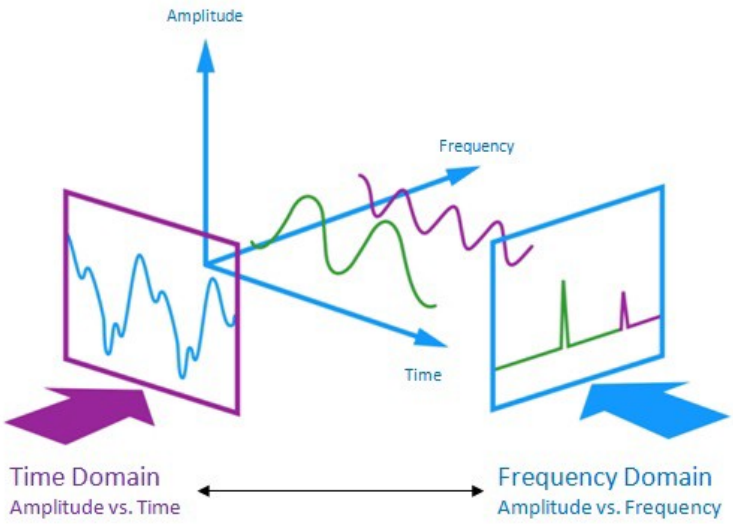

Figure 2.2: Representation of an audio signal in the time and frequency domains [11].

Historically, using the raw audio signal, in the time domain, as input to predictive models often was not adequate to achieve good performance. Instead, specific features of the signal were extracted and used in the modelling process. There are a range of different features which can be extracted from an audio signal and each feature conveys different information about the signal. Feature extraction and selection can be a tedious and labour-intensive task which requires great domain knowledge. With the advent of deep learning, sophisticated feature extraction methods were no longer required to achieve good results. The network, if properly constructed and trained, could learn to extract the most vital features from the data by itself. With enough quality data, there often is no need to extract any audio features. It is relatively standard however to extract and use a few different audio features in order to aid the network in the learning process and speed up the training. Popular audio features used in deep learning models include the spectrogram, mel spectrogram, and the mel frequency cepstral coefficients (MFCCs).

In order to shift the signal from the time domain to the frequency domain, a Fourier transform can be applied to the time series. The Fourier transformed time signal reveals information regarding the power and phase of the frequencies of the sinusoidal waves present in the signal. Letting $\{x(n), n = 0, 1, 2, \ldots, N-1\}$ be a sequence of real valued data samples, then the Fourier transform, here denoted $X(f)$, in discrete time, can be computed according to equation 2.1 [12]. This is the Discrete Time Fourier Transform (DTF) applied to a finite sequence of data.

$$X(f) = \sum_{n=0}^{N-1} x(n) \cdot e^{-\frac{2\pi i}{N} kn} = \sum_{n=0}^{N-1} x(n) \cdot \left[\cos\left(\frac{2\pi}{N} kn\right) - i \cdot \sin\left(\frac{2\pi}{N} kn\right) \right] \tag{2.1}$$

The non-discrete Fourier transform assumes that the time signal is continuous in nature and therefore infinite in length. The transition to discrete time results in certain distortions to the output called spectral leakages (side lobes). Side lobes present themselves as a loss of detail in the output of the DFT. The main lobes of the spectral representation are therefore more difficult to distinguish in the presence of side lobes. Digital signals are, naturally, not infinite in length and so applying the DFT to digital audio signals will lead to a lower resolution of the output. In order to remedy this flaw, so called window functions can be used. A window function is a non-rectangular function applied to the time signal along certain pre-determined steps. The choice of window function can impact the height of the main lobes as well as the width of the side lobes, and should be chosen according to the application. In this thesis, the Hanning window function is used, and can be calculated according to $h(t) = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{2\pi t}{n-1}\right),\ t = 0, \ldots, N-1$ [12]. The signal is windowed by multiplying it with the output from the Hanning function where N denotes the length of the window. The power spectral density (PSD) of the signal can be represented with the periodogram. The periodogram is the squared magnitude of the Fourier transform formed as equation 2.2 [12].

$$\hat{R}_X(f) = \frac{1}{N}|X(f)|^2 \tag{2.2}$$

The DFT algorithm is relatively computationally inefficient. As the number of samples, N, grows large, the computational complexity grows at a rate of $O(N^2)$. Replacing the DFT with the Fast Fourier Transform (FFT) reduces the computational complexity to $O(N\log(N))$. There are several variations of the FFT, all with complexities $O(N\ \log(N))$, but the most common variant is the radix-2 decimation in time (DIT) algorithm by Cooley and Turkey [13]. The idea behind the radix-2 DIT algorithm is to decompose the DFT into several smaller transformations, thereby reducing the number of computations required. The algorithm decomposes the N-point time signal recursively into N different individual time domain signals, each composed of a single value. The first step is to split the sequence into even and odd indexed inputs $(x_{2m} = x_0, x_2, \ldots, x_{N-2})$ and $(x_{2m+1} = x_1, x_3, \ldots, x_{N-1})$, respectively, and compute the DFT of these two sequences. These can then be combined in order to produce the final DFT of the whole input sequence. The algorithm therefore decomposes the original equation 2.1 into

$$X(f) = \sum_{n=0}^{N/2-1} x(2n)e^{-\frac{2\pi i}{N}(2n)f} + \sum_{n=0}^{N/2-1} x(2n+1)e^{-\frac{2\pi i}{N}(2n+1)f} \tag{2.3}$$

This procedure can then be performed recursively on the whole sequence according to

algorithm 1.

Algorithm 1: Fast Fourier Transform (FFT)

Data: $x = \{x_0, x_s, x_{2s}..., x_{(N-1)s}\}$
N is a number to the power of 2
s is the stride of the input x
Result: X = $\{X_0, X_1, ..., X_{N-1}\}$
begin
  if $N = 1$ **then**
    $X_0 = x_0$
  else
    $X_{0,...,N/2-1} = FFT(x, N/2, 2s)$
    $X_{N/2,...,N-1} = FFT(x + s, N/2, 2s)$
    for $k = 0$ ***to*** $N/2 - 1$ **do**
      $t = X_k$
      $X_k = t + e^{-2ik/N} X_{k+N/2}$
      $X_{k+N/2} = t - e^{-2ik/N} X_{k+N/2}$
    end
end

When using the Fourier transform, the signal is assumed to be stationary in nature, meaning that the mean and variance of the signal should be constant with respect to time. Particularly for longer recordings, it is natural to assume that this may not hold true. During the analysis of recordings, it is therefore necessary to divide the signal into smaller "frames" for which, during the smaller time periods, we can assume stationarity to hold true. Every frame typically covers around 20-40 ms each, and depending on the sampling rate, contains around 200-2000 samples. The frames can then be transformed into the frequency domain using the DTF, or a variant such as the FFT. Using the transformed frames, we can gauge how the frequency spectrum progresses throughout the recording. The algorithm used for dividing the recording into frames and transforming these into the frequency domain is called the Short Time Fourier Transform (STFT). With $w(n)$ as the specified window function, $x(n)$ the signal, H the hop length, and N the window length. The STFT, in discrete time, is formed as

$$\mathbf{STFT}\{x(n)\}(m,k) \equiv X(m,k) = \sum_{n=0}^{N-1} x(n+mH)w(n)[n-m]e^{\frac{-2\pi i}{N}kn} \tag{2.4}$$

In other words; the complex number $X(m,k)$ denotes the k^{th} Fourier coefficient for the m^{th} time frame. Combining the magnitude squared of all the frame level frequency spectra for a recording yields the spectrogram for that recording. The spectrogram is defined according to

$$\text{Spectrogram}\{x(n)\}(m,k) \equiv |X(m,k)|^2 \tag{2.5}$$

Figure 2.5 displays an example of two spectrograms computed for two distinct audio recordings.

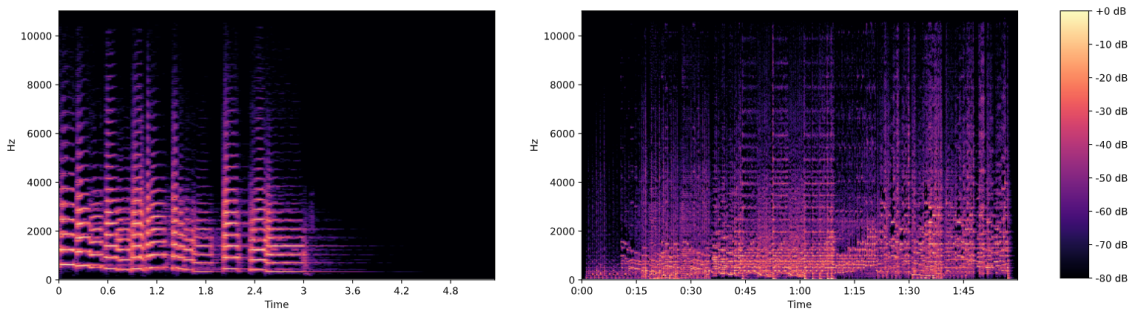

Figure 2.3: Spectrograms of two sample audio recordings.

The cochlea, a part of of the human ear which interprets sound waves, cannot detect differences in closely spaced frequencies. This phenomenon gets progressively more pronounced towards higher frequencies. Translating this to the spectrogram, we can transition from the frequency scale to the mel scale in order to make it more representative of human hearing. The mel scale is the result of a non-linear transformation of the frequency scale, altering the pitch distances depending on what part of the frequency spectrum they are located. In order to transform the spectrogram to mel scale, a mel-spaced filterbank (mel filterbank) can be applied to the data. A mel filterbank is an arbitrary number of different triangular filters which, when applied to the spectrogram provide an indication of the energy available at different frequency bands. These bands are of varying sizes depending on what part of the frequency spectrum they are located. Bands at lower frequencies cover a smaller range of the spectrum and bands at higher frequencies cover a larger range. An example of a filterbank with 40 filters can be seen in Figure 2.5. Oftentimes, the log of these energies are then computed, yielding the log filterbank energies. Figure 2.4 displays the mel spectrogram for the same two recordings from Figure 2.3.

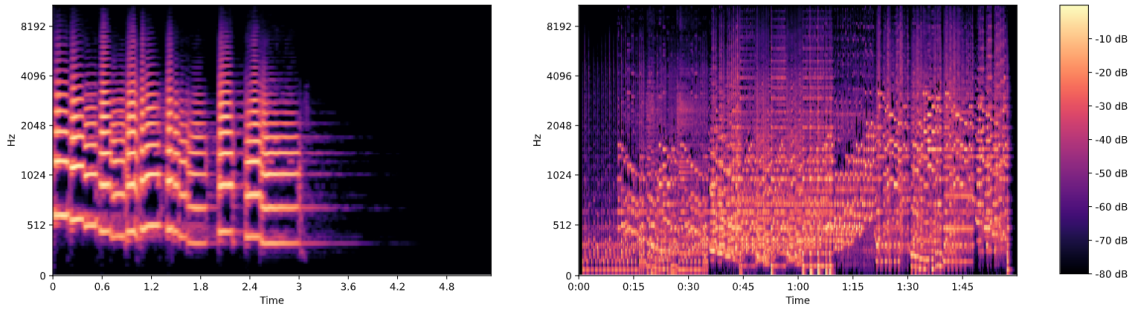

Figure 2.4: Mel spectrograms of the two sample audio recordings in Figure 2.5.

Another commonly used set of audio features are the Mel Frequency Cepstral Coefficients (MFCCs). The MFCCs can be obtained by transforming the mel spectrogram using a discrete cosine transformation (DCT) given by equation 2.6, where B_{pq} is the DCT coefficient of the input matrix A with the dimensions MxN. Figure 2.6 displays en example of two MFCC representations of the two same sounds utilised in previous examples.

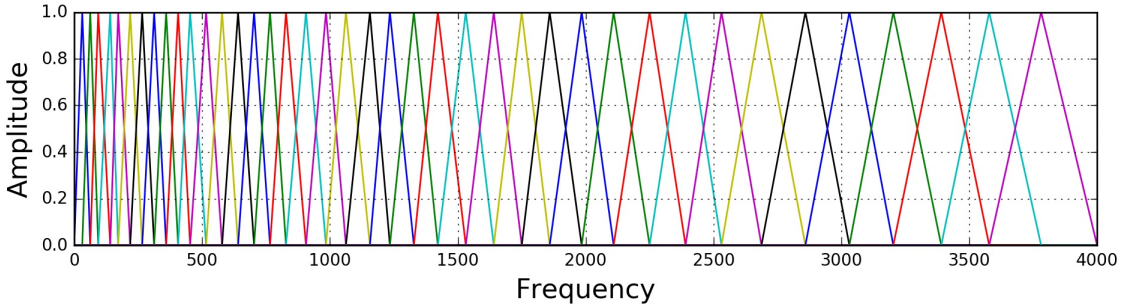

Figure 2.5: Mel filterbanks takes from [14]

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \quad \begin{matrix} 0 \leq p \leq M-1 \\ 0 \leq q \leq N-1 \end{matrix}$$

$$\alpha_p = \begin{cases} 1/\sqrt{M}, & p = 0 \\ \sqrt{2/M}, & 1 \leq p \leq M-1 \end{cases} \quad \alpha_q = \begin{cases} 1/\sqrt{N}, & q = 0 \\ \sqrt{2/N}, & 1 \leq q \leq N-1 \end{cases} \tag{2.6}$$

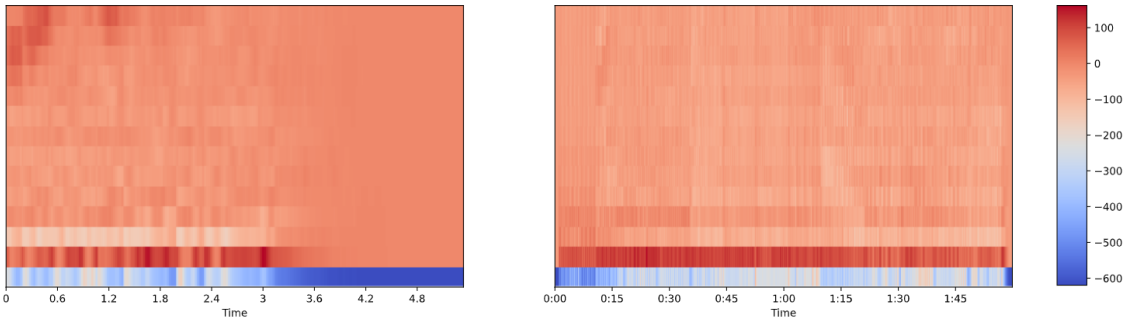

Figure 2.6: MFCCs of the two sample audio recordings in Figure 2.5.

The process of extracting the aforementioned features can be seen in Figure 2.7.

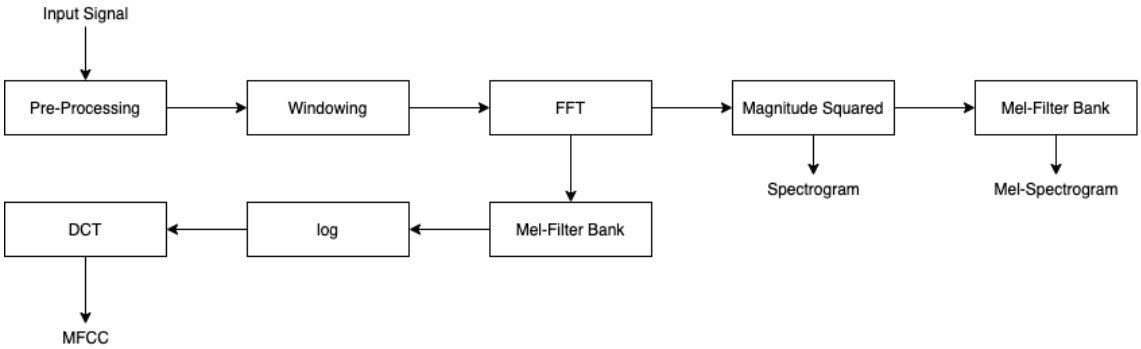

Figure 2.7: The process of extracting the spectrogram, mel spectrogram, and MFCCs from a recorded input signal.

The harmonic to noise ratio (HNR) is a measure of the ratio between periodic versus non-periodic components in a time signal, and is defined as $\frac{\sum_{k=0}^{N-1} h[k]^2}{\sum_{k=0}^{N-1} n[k]^2}$ where $h[k]$ is the harmonic component, and $n[k]$ is the noise component of the signal. The metric can

be used to detect the proportion of energy of the sound waves which come from the harmonics, and the proportion which comes from the noise. The ratio can therefore be used as a measure of the signal-to-noise ratio of a periodic signal. The HNR is, in this study, applied to each frame of the recordings and used as input to the machine learning models.

Autoregressive (AR) models are a type of model used to represent random processes. As such, they are popular within speech modelling. The output of an AR model is linearly dependent on its previous values and an additional stochastic term. Letting $A(z)$ be a stable polynomial of degree p defined as $A(z) = a_0 + a_1 z + \cdots + a_p z^p$. A stationary stochastic sequence $\{X_t\}$ is called an $AR(p)$-process with generating polynomial $A(z)$, if the sequence $\{e_t\}$, given by

$$X_t + a_1 X_{t-1} + \cdots + a_p X_{t-p} = e_t \tag{2.7}$$

is a white noise sequence, which is uncorrelated with $X_{t-1}, X_{t-2}, \ldots$. The variables e_t are innovations to the AR-process. If we know the autocovariance of the process, the $AR(p)$-parameters can be solved through the Yule Walker equations

$$\begin{gathered} r_X(k) + a_1 r_X(k-1) + \cdots + a_p r_X(k-p) = 0, \text{ for } k = 1, 2, \ldots \\ r_X(0) + a_1 r_X(1) + \cdots + a_p r_X(p) = \sigma^2, \text{ for } k = 0 \end{gathered} \tag{2.8}$$

where σ is the covariance of the white noise sequence $\{e_t\}$.

The pitch is yet another feature which can be calculated from the time signal. The pitch, or fundamental frequency, of a recording can be calculated in a number of ways. In this thesis, the YIN-algorithm [15], which extends the autocorrelation method, is used to calculate the pitch of every frame. In order to smooth out the pitch across frames, the pitch estimates are passed through a low pass filter. We can also observe the pitch progression across individual recordings through fitting a linear equation (regression) to the pitch and extracting the coefficients of that model.

Furthermore, the amplitude of the overtones, the so called tonal amplitudes, are also extracted from the audio signal.

The jitter is a measure of the deviation from the true periodicity of a periodic signal. This can be measured in a variety of manners. In this thesis, we use the absolute jitter (jitta), local jitter (jitt), relative average pertubation (RAP), and the five-point period perturbation quotient (PPQ5) as indications of the jitter. The absolute jitter (jitta) of a signal is the average absolute difference between the consecutive periods, and can be calculated according to $\frac{1}{N-1}\sum_{i=1}^{N-1} |T_i - T_{i-1}|$ where T_i is the glottal period length and N is the number of glottal periods [16].

The local jitter is the average absolute difference between the consecutive periods divided by the average periods, given as a percentage, i.e.,

$$\text{jitt} = \frac{\frac{1}{N-1}\sum_{i=1}^{N-1}|T_i - T_{i-1}|}{\frac{1}{N}\sum_{i=1}^{N} T_i} * 100 \tag{2.9}$$

The RAP is a measure of the average absolute difference between a period and the average of it combined with its two closest neighbors, divided by the average period, and as a percentage, i.e.,

$$\text{RAP} = \frac{\frac{1}{N-1}\sum_{i=1}^{N-1}|T_i - \frac{1}{3}\sum_{n=i-1}^{i+1} T_n|}{\frac{1}{N}\sum_{i=1}^{N} T_i} * 100 \tag{2.10}$$

The PPQ5 is the average absolute difference between a period and the average of it combined with its four closest neighbors, divided by the average period, and as a percentage, i.e.,

$$\text{PPQ5} = \frac{\frac{1}{N-1}\sum_{i=2}^{N-2}|T_i - \frac{1}{5}\sum_{n=i-2}^{i+2} T_n|}{\frac{1}{N}\sum_{i=1}^{N} T_i} * 100 \tag{2.11}$$

Shimmer is a measure of how much the amplitude is changing across periods in a periodic signal. Four different estimates for the shimmer which are used in this thesis. They are the absolute shimmer (ShdB), local Shimmer (Shim), 3-point Amplitude Perturbation Quotient (APQ3), and 5-point Amplitude Perturbation Quotient (APQ5). The absolute shimmer, in decibels, is the average absolute log of the ratio between amplitudes of consecutive periods multiplied by 20, such that

$$\text{ShdB} = \frac{1}{N-1}\sum_{i=1}^{N-1}\left|20 * \log\left(\frac{A_{i+1}}{A_i}\right)\right| \tag{2.12}$$

with A_i being the amplitude and N being the number of glottal periods.

The local shimmer is the average absolute difference between amplitudes of consecutive periods divided by th average amplitude, such that

$$\text{Shim} = \frac{\frac{1}{N-1}\sum_{i=1}^{N-1}|A_i - A_{i+1}|}{\frac{1}{N}\sum_{i=1}^{N} A_i} * 100 \tag{2.13}$$

The 3-point Amplitude Perturbation Quotient is the average absolute difference between the amplitude of a period and the average of the amplitude of that period and the two closest neighbors, divided by the average amplitude, i.e.,

$$\text{APQ3} = \frac{\frac{1}{N-1}\sum_{i=1}^{N-1}\left|A_i - \left(\frac{1}{3}\sum_{n=i-1}^{i+1} A_n\right)\right|}{\frac{1}{N}\sum_{i=1}^{N} A_i} * 100 \tag{2.14}$$

The 5-point Amplitude Perturbation Quotient is the average absolute difference between the amplitude of a period and the average of the amplitude of that and the four closest neighbors, divided by the average amplitude, i.e.,

$$\text{APQ5} = \frac{\frac{1}{N-1}\sum_{i=2}^{N-2}\left|A_i - \left(\frac{1}{5}\sum_{n=i-2}^{i+2} A_n\right)\right|}{\frac{1}{N}\sum_{i=1}^{N} A_i} * 100 \tag{2.15}$$

A representation of the jitter and shimmer of a recording can be seen in Figure 2.8. The glottal period referenced above is the time period between consecutive peaks, and the glottal amplitude is the height of the peaks.

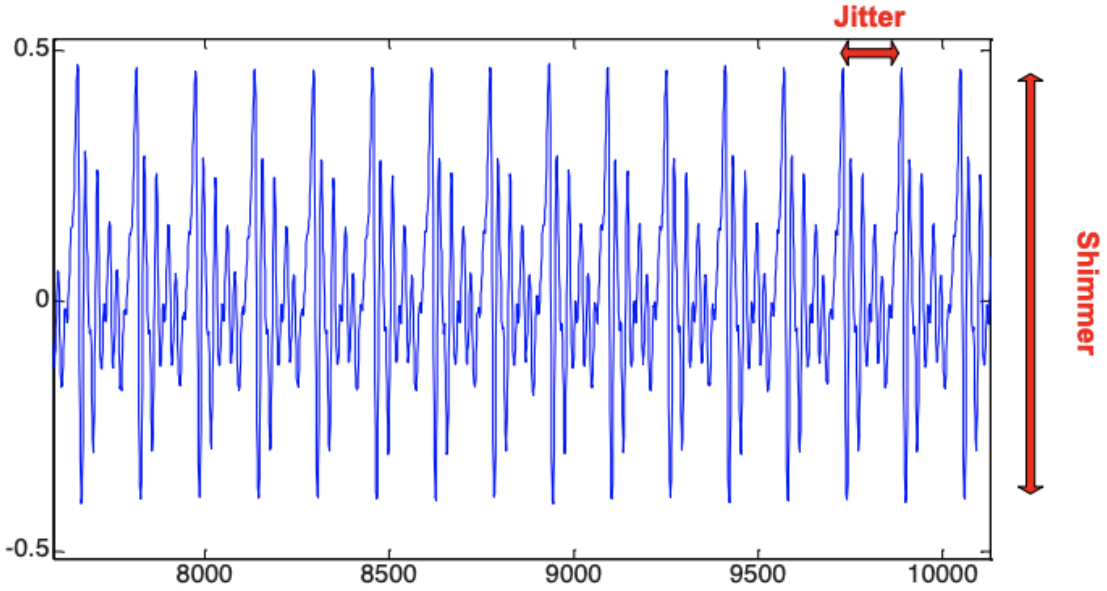

Figure 2.8: Depiction of the jitter and shimmer as seen in the signal of a recording [17].

2.2 Machine Learning

Machine Learning can, in broad terms, be defined as the study of algorithms which improve through experiences. In other words; algorithms that can learn. Machine Learning is an umbrella term which encompasses many sub-categories, with the most relevant category for this thesis being supervised learning. In supervised learning, the algorithm learns a specific function which maps the input to an associated output. The input fed to the model can, for example, be an array or an image, and the output can be a number or label. Through feeding the algorithm many such input-output pairs, and letting the model learn from its mistakes and successes, one is able to create a model which can, hopefully, perform well on never before seen inputs.

Unsupervised machine learning on the other hand encompasses models which can learn patterns from unlabeled data. As such, through mimicry, the models are able to build an internal representation of the world which can be used to classify and cluster inputs.

Unsupervised learning is used in this thesis in the form of the t-SNE algorithm, which transforms higher dimensional data to a lower dimensional representation.

Depending on the goal of the model, it can be trained for either classification or regression tasks. The former entails that the outputs are divided into distinct groups whilst the latter involves a continuous output. Given an input, classification models predict labels whilst regression models predict quantities. The most simple form of classification is called binary classification. In binary classification problems, there are two distinct groups which the input can be mapped to. For example, a model which predicts if a specified input image is a cat or a dog, is a binary classifier.

In most machine learning projects, the data (input-output pairs) is typically divided into a training set and a test set. The training set is used to train the model whilst the test set is used to test the performance of the model after it has been trained. Alternatively, the data can be divided into three distinct sets with the third, in this case, being the validation set which serves a similar purpose to the test set, however it is used throughout the training process to check for overfitting of the model.

2.2.1 T-distributed Stochastic Neighbor Embedding (t-SNE)

T-distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised machine learning algorithm first presented in the paper "Visualizing Data using t-SNE" [18]. The method can be used to transform higher dimensional data into a representation in lower dimensions, while preserving both the local and global structure of the data. Figure 2.9 depicts an example where the algorithm has been applied to the MNIST dataset, with the final representation being in two dimensional space. In this example, each point in 2D space represents an image with original dimensions of 28x28 pixels, and therefore 784 distinct features. The algorithm is able to separate the classes with little overlap between classes. The distances in 2D space are well preserved as, for example, images with digits 9 and 4 are very close to each other, just as they perceived to humans.

The t-SNE algorithm constructs joint probability distributions for all pairs of points in high-dimensional space, as well as joint probability distributions for all pairs of points in the low-dimensional space. The distance between the pairs in the respective distributions are then minimized through minimizing the Kullback–Leibler divergence (KL divergence).

The conditional probabilities between nearby points in the higher dimensional space can be computed according to equation 2.16, where x_i and x_j are two points in the high dimensional space and σ_i is the variance of the Gaussian that is centered on datapoint x_i. This conditional probability is therefore represented as a Gausian distribution centered at x_i with a standard deviation of σ_i.

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{2.16}$$

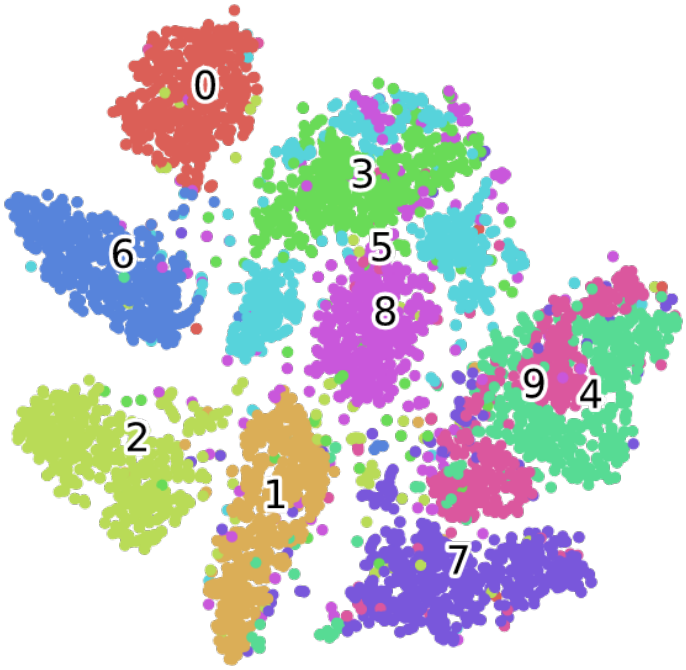

Figure 2.9: Visualization produced by the t-SNE algorithm run on the MNIST dataset.

The joint probability distributions can then be calculated, using the conditional probabilities from above, as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{2.17}$$

The conditional probabilities between nearby points in the lower dimensional space can be calculated as $\frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq i} exp(-||y_i - y_k||^2)}$. This time, the joint probability distributions are created using the t-distribution, as opposed to the Gaussian. The heavy tails of the t-distribution will lead to less crowding in the lower dimension.

The gradient used to update the low dimensional representation $Y^{(T)}$, is the gradient of the KL divergence between the Gaussian and student-t distributions. It can be derived as

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{2.18}$$

A general overview of the algorithm can be seen in algorithm 2 below.

Algorithm 2: t-distributed Stochastic Neighbor Embedding (t-SNE)

Data: $X = \{x_1, x_2, ..., x_n\}$
cost function parameters: perplexity $Perp$
optimization parameters: number of iterations T, learning rate η,
momentum $\alpha(t)$
Result: Low dimensional(2D or 3D) data representation
$Y^{(T)} = \{y_1, y_2, ..., y_n\}$
begin
  compute pairwise affinities $p_{i|j}$ with perplexity $Perp$ (using Equation 2.16)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $Y^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for $t = 1$ ***to*** T **do**
    compute low-dimensional affinities q_{ij}
    compute gradient $\frac{\delta C}{\delta Y}$ (using Equation 2.18)
    set $Y^{(t)} = Y^{(t-1)} + \eta \frac{\delta C}{\delta Y} + \alpha(t)(Y^{(t-1)} - Y^{(t-2)})$
  end
end

2.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are a class of computing systems which can be used for a range of different problems such as classification and regression. With today's access to cheap computing power and large amounts of data, ANNs have become very popular and widely available. ANNs are, much like the name indicates, inspired by the function of *neurons* in the brain. Just as the brain contains neurons and synapses in order to transmit electric impulses along neuronal pathways, ANNs contain interconnected layers of artificial neurons which can relay signals throughout the network. Signals, in the form of an input, enter the network at one end and propagate throughout the network towards the other end, where an output is produced. Figure 2.10 illustrates an example of an ANN with three hidden, also called dense, layers.

The output of each neuron can be modeled with a simple equation combining the constituent inputs to the neuron. Looking at a single neuron from Figure 2.10, it can be represented as in Figure 2.11, with inputs $x_1, x_2, ..., x_n$, weights $w_1, w_2, ..., w_n$ and bias term b. In all essence, each neuron in the network computes a weighted sum of its inputs, applies an activation function to the sum, and then outputs the result to neurons in the next layer of the network.

The activation function is applied to the output at each neuron in order to enable the network to learn more complex functions. Through the addition of a non-linear activation function, the network is able to learn non-linear function as opposed to solely linear boundaries. It achieves this by acting as a gate which regulates the neuronal

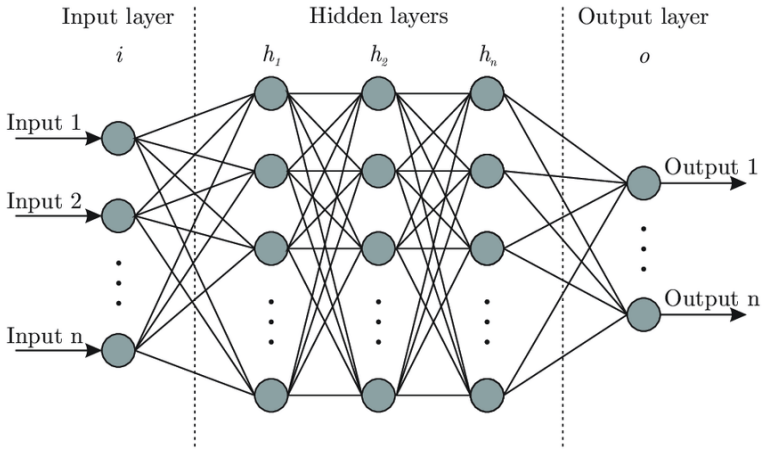

Figure 2.10: ANN with three hidden layers, n inputs, and n outputs [3].

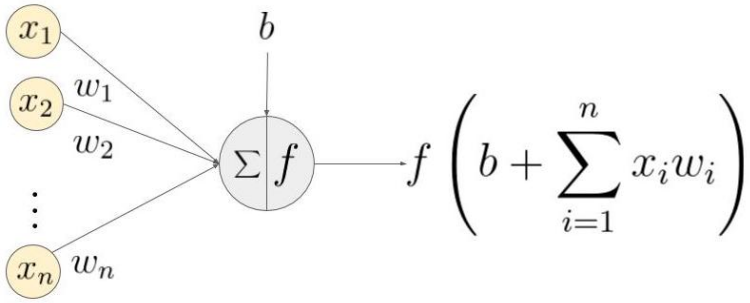

Figure 2.11: Mathematical operation performed in a single neuron of an ANN [19].

output to the next layer. Commonly used activation functions include ReLU, Tanh, Sigmoid, Linear, and Softmax. These can be seen in Figures 2.12 through 2.15.

In order for the network to learn the optimal function mapping of the given inputs to their respective outputs, the weights of the network need to be updated. This is done through a process called backpropagation. Backpropagation trains the network by updating its weights, one step at the time, through minimizing a given loss function. A loss function is a pre-defined metric with which we evaluate the performance of a model. Given the predicted output $\hat{y}$, and the ground truth y, the loss function can be calculated according to a resulting formula. There are several different variations of loss functions which have different uses depending on the task. For classification tasks, the Binary Cross Entropy (BCE) or Categorical Cross Entropy (CCE) loss functions are very popular choices. As will be apparent later in this thesis, standard Siamese Networks perform a variation of the binary classification task, and therefore the BCE loss function can be utilized. The BCE can be computed as $-\frac{1}{N}\sum_{i=1}^{N}(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$. Using this expression, it is easy to see that the loss is minimized when $\hat{y} = y$; in other words when the predicted output is the same as the ground truth. The BCE cannot take on a value of zero unless both $\hat{y}$ and y are equal to zero.

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)) \tag{2.19}$$

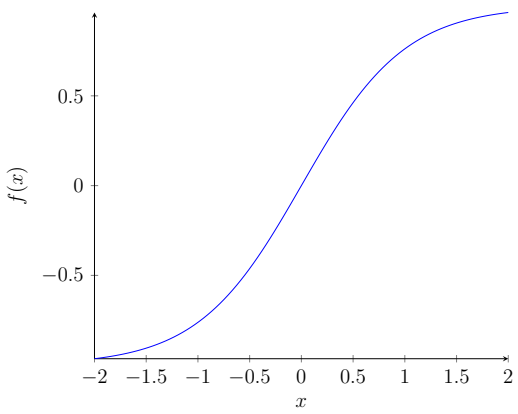

Figure 2.12: Tanh activation function

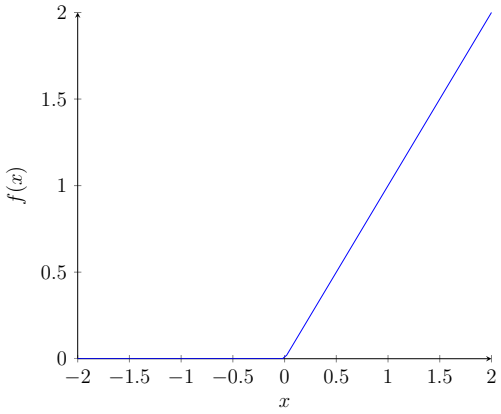

Figure 2.13: ReLu activation function

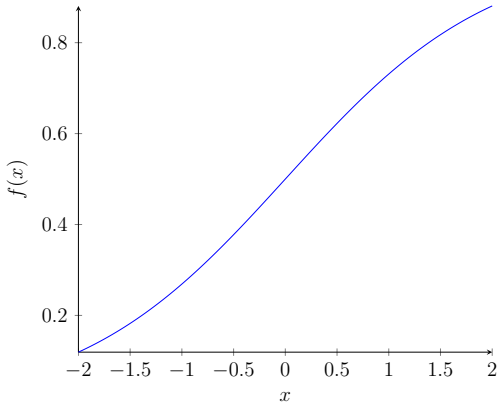

Figure 2.14: Sigmoid activation function

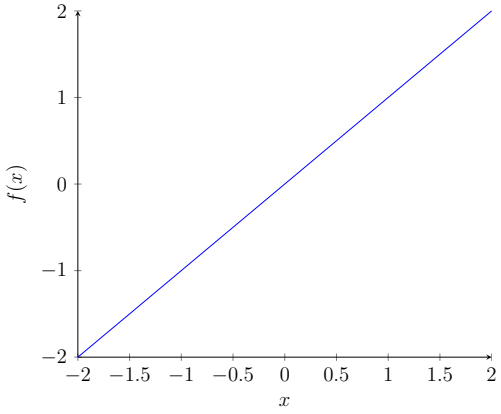

Figure 2.15: Linear activation function

After all of the inputs in the dataset have been propagated forwards through the network, and the loss has been calculated for each one of these inputs, the average loss is calculated and and used to update the weights of the network. In order to minimize the loss obtained at the next iteration, we take the derivative of the average loss with regard to the weights and move the weights in the negative direction of the gradient according to $\theta^{t+1} = \theta^t + l \bullet \frac{\partial L(X,\theta^t)}{\partial \theta}$, where l is the learning rate of the network, L is the loss function, and θ^t are the weights and biases of the network at iteration t of the algorithm. This is the idea behind the Gradient Descent (GD) algorithm, which lays the foundation for modern machine learning.

As opposed to updating the weights using the whole dataset, either single instances, or batches of inputs can be sent to the model. In the case of single instances, the weights are updated after each forward propagation, using the loss associated with individual instances. When using batches of inputs, the average loss for said batch is used to update the network. With single instance updates of the network, the inputs are often sampled randomly from the dataset, which is the reason why this variation of the GD algorithm is called Stochastic Gradient Descent (SGD). On the other hand, if batches are used to update the network, the algorithms appropriately takes the name of Mini-Batch Gradient Descent; a more stable version of the SGD algorithm with

regard to convergence.

The gradient of the loss function, $\frac{\partial L(X,\theta^t)}{\partial \theta}$, is calculated starting at the end of the network, and then the loss is propagated backwards throughout the network until the start is reached, upon which the parameters of the network are updated. The derivation of the loss function with regards to the weights at the different layers of the network can be performed using the chain rule

2.4 Convolutional Layers

Dense layers can be replaced with other forms of intermediate network layers. Convolutional layers are another example of a type of layer used for feature extraction. Convolutional layers are especially proficient at extracting specific patterns from images, even as the location of the pattern within the image changes. These layers work by applying a filter to the input in order to create a feature map containing the extracted features. The feature map is created by taking a filter/kernel in the form of a matrix and sliding it over the image, calculating the output at every step of the process. The feature map values are calculated as $G[m,n] = (f \bullet h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k]$, where f is the input image and h is the kernel/filter. The indices of rows and columns of the matrix $G[m,n]$ are here named m and n respectively.

Figure 2.16 depicts an example of the convolutional operations performed in each convolutional layer. The result at every step of the filter is calculated as the dot-product between the filter and the image, followed by the summation of the result. Depending on the type of filter which is applied to the image, the output feature map will pick up and accentuate different features.

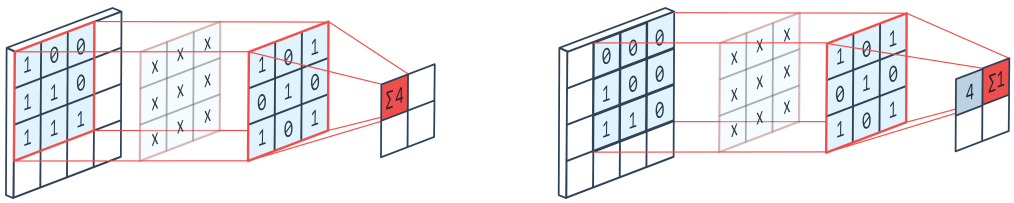

Figure 2.16: Convolutional operation with 2D input and filter [20].

As seen in Figure 2.16, the resulting feature map is oftentimes smaller than the input image. As a result, this operation can only be performed so many times before the image disappears. This can be remedied by padding the input image with zeros at each layer. Furthermore, the stride of the filter can be altered in order to change the degree of overlap of the kernel at each step. All these parameters can, and should, be altered depending on the dimensions of the image and the type of problem faced with.

The same principles can be extended to convolutions in the first and third dimensions. With one dimensional data, the kernel is also one dimensional. We must then choose the size of the kernel which is to be mapped to the input. The kernel slides in one direction, from left to right, over the data, as seen in Figure 2.17

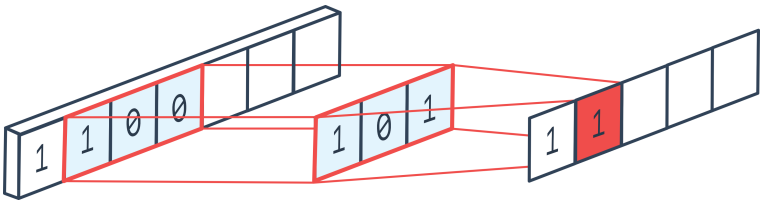

Figure 2.17: Convolutional operation with 1D input and filter [21].

2.5 Autoencoders

An autoencoder is a type of ANN which aims to learn an efficient representation (encoding) of the specified input data. Using this architecture can greatly reduce the dimensionality of the data. Through gradually reducing the size of the layers of the network, an autoencoder compresses the information to a lower dimensional latent space representation. This can be seen in Figure 2.18 where the red layer of the network represent the new, and smaller, encoding of the original input. Autoencoders oftentimes pair an encoder with a decoder in an architecture such as the one seen in Figure 2.18, where the blue layers represent the encoder, and the yellow layers represent the decoder. This enables the network to learn to reconstruct the input from the encoded data, in an unsupervised manner.

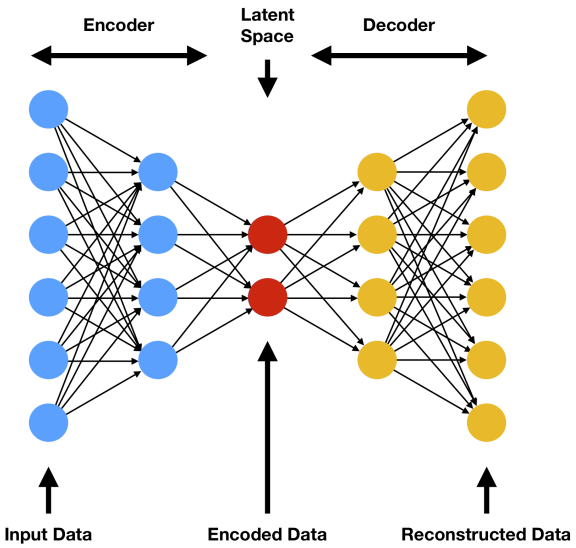

Figure 2.18: Autoencoder structure containing encoder (blue), latent space vector (red), and decoder (yellow) [22].

In this thesis, the encoder structure is used to reduce the size of the feature vector used to distinguish between inputs. The assumption is that the network learns to extract the most important features of the input and represent them in the encoding. Only the encoder portion of the autoencoder structure is used as we do not wish to reconstruct the encoded data after it has been compressed. This also makes the learning process supervised, as opposed to the unsupervised learning which, is often used when training autoencoders.

2.6 N-Shot Learning

Regular ANNs generally require large amounts of data to train and maintain. Additionally, in the case of supervised learning, that data needs to be labeled. Labeling the data is oftentimes costly and can be a lengthy process, and something not all machine learning practitioners have at their disposal. Furthermore, comparing machine learning to human learning, we recognise that machines are significantly less efficient in their learning process. The large amounts of data required for a machine to learn is not comparable to the few instances a human requires before they can learn to recognise features in objects. For example, most machine learning algorithms require thousands of images of new instances in order to recognise what a certain class looks like, whereas humans would only need a few.

In the past few years, much research has therefore been dedicated to the field of few-shot learning. Few-shot learning is the sub-set of machine learning which deals with building models and architectures which can classify data with only one, or a few, data instances available from each class during training. For example, say we were to build a machine learning model to predict the species associated with an inputted image of a bird. Some bird species are very rare and thus we may only have access to a few, or even a single image of said bird. This then becomes a one-shot learning problem since we want our model to be able to classify that species correctly in the future. Siamese Neural Networks are a type of network architectures which have proven themselves highly proficient at few shot learning tasks.

2.7 Siamese Neural Networks

In cases when data is scarce, Siamese Neural Networks (SNN) are a great tool at achieving good results for certain machine learning tasks, classification being one of them. SNNs take a different approach to classification over regular ANN architectures. Where as ANNs learn class-specific features, SNNs learn to distinguish between classes of inputs through the computation of similarity scores. A lower similarity score implies lower chances of input pairs originating from the same class and vice versa. For example, a SNN can be trained to detect similarities between images of faces, and then used to predict whether a pair of images containing faces originate from the same individual or not.

Siamese Neural Networks, or Siamese Networks for short, make use of two or more branches of identical networks which are run in tandem with each other, both during training and testing. Identical in this case implies that the networks have the same structures, weights, and parameters. The parameters are updated simultaneously and mirrored across all sub-networks. Each branch of the network accepts an input, such as an image or a 1D vector of features. The network computes a "fingerprint", called an embedding, for every one of these inputs. This embedding is produced by the encoder structure shown in Figure 2.18. Training the network serves as an optimisation of the latent space of the encoder in order for inputs from differing classes to produce differing embeddings, and inputs from same classes to produce similar embeddings.

The similarity of embeddings can be calculated using metrics such as the euclidean distance. SNNs therefore learn an embedding space which is highly generalisable on new classes and their unknown distributions.

Problems such as bias often arise when the dataset used to train the models is unbalanced with regards to the number of instances available in every class. There are a number of ways to remedy an imbalanced dataset such as custom loss functions, the generation of synthetic data, and custom data sampling. However, imbalanced datasets are not a problem for SNNs. This is a major advantage of these models.

In the groundbreaking paper by Koch et al [5], which popularised SNNs, they trained a SNN on the Omniglot dataset. The Omniglot dataset is a dataset containing hand-drawn letters from 50 different alphabets, with only 20 distinct instances of each class (letter). The lack of data for each class makes it a difficult problem to tackle with regular ANNs. Their convolutional SNN architecture achieved a one-shot accuracy of 92%, which at the time was second only to that of Hierarchical Bayesian Program Learning algorithms and humans. As we will see throughout this thesis, the use cases of the technique extends far beyond differentiation between letters in alphabets.

A key benefit of using SNNs as opposed to regular ANN architectures is the increased robustness of the network. "Once a network has been tuned, we can then capitalise on powerful discriminative features to generalize the predictive power of the network not just to new data, but to entirely new classes from unknown distributions" [5]. This eliminates the need to retrain the network once new classes are encountered and decreases the overall need for data.

The choice of ANN architecture chosen for the sub-network is arbitrary. The sub-network can for example combine convolutional layers with dense layers to achieve good results on image classification tasks. Figure 2.19 depicts what a SNN could look like with 2 sister networks (sub-branches), and a simple single dense layer architecture as sub-networks. The embedding is the vector output from each branch of the network. This embedding is the vector representation of the input which the network produces. The distance metric compares the distance between these embeddings in order to assess the distance between them.

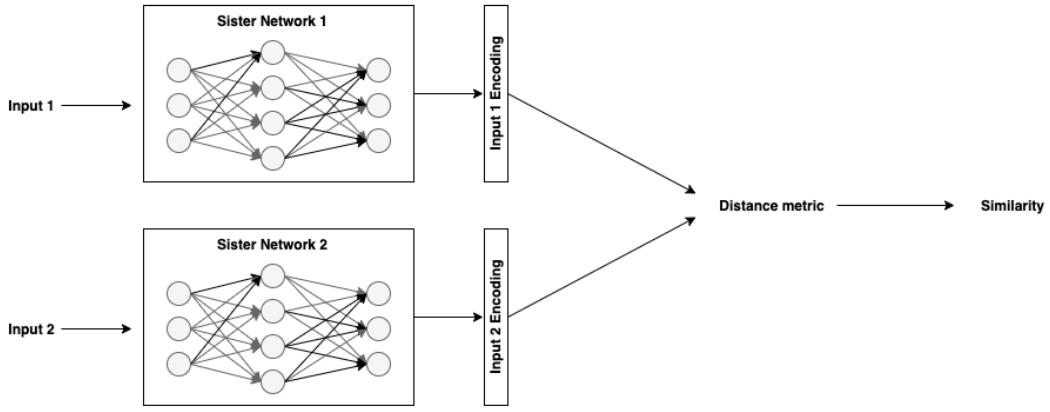

Figure 2.19: Structure of a SNN.

The distance metric computed using the embeddings can be re-scaled to values between

zero and one. Passing the distance through the sigmoid function, presented in Figure 2.14, will produce a more interpretable result.

There are a few different metrics which can be used to measure the distance, in vector space, between pairs of embeddings produced by the networks. The first, and perhaps most common approach, is the euclidean distance given as

$$d_e(p,q) = \sqrt{\sum_{i=1}^{n} \left(|p_i - q_i|\right)^2} \tag{2.20}$$

with p and q in this case being the vectors of comparison, and n being the length of said vectors.

Secondly, the Manhattan distance is defined as

$$d_m(p,q) = ||p - q|| = \sum_{i=1}^{n} \left(|p_i - q_i|\right) \tag{2.21}$$

where again, p and q are the vectors we wish to compare.

The cosine distance is also a popular metric used to assess the distance between two vectors. It is defined as the cosine of the angle, in vector space, between the two inputs, i.e.,

$$d_c(p,q) = 1 - \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}} \tag{2.22}$$

There are several possible metrics used to compute the distance between matrices. The similarity measure used in this thesis is that of the Frobenius norm, which can be calculated as

$$d_{mat}(A,B) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(a_{ij} - b_{ij})^2} \tag{2.23}$$

with A and B being the input matrices.

The SNNs learn through backpropagation and gradient descent algorithms just like other ANNs do. Parameter updates in SNNs occur across both networks simultaneously and identically. The derivative of the loss with regard to the parameters in the twin-networks produce the same set of values across both networks, thus leading to the sister-networks remaining identical throughout iterations.

2.8 Alternate Siamese Architectures and Loss Functions

Alternative versions of SNNs, with differing structures, have been proposed to further improve on the original Siamese architecture. A structure which has seen good performance within few-shot learning tasks is the Siamese Network with three sub-networks and a triplet loss function. This triplet network compares an anchor input with both a negative (pairs from differing classes), and a positive (pairs from same classes) input. The loss is calculated based on these two pairwise distances. An example of the triplet network with a simple dense ANN as sub-network architecture can be seen in Figure 2.20.

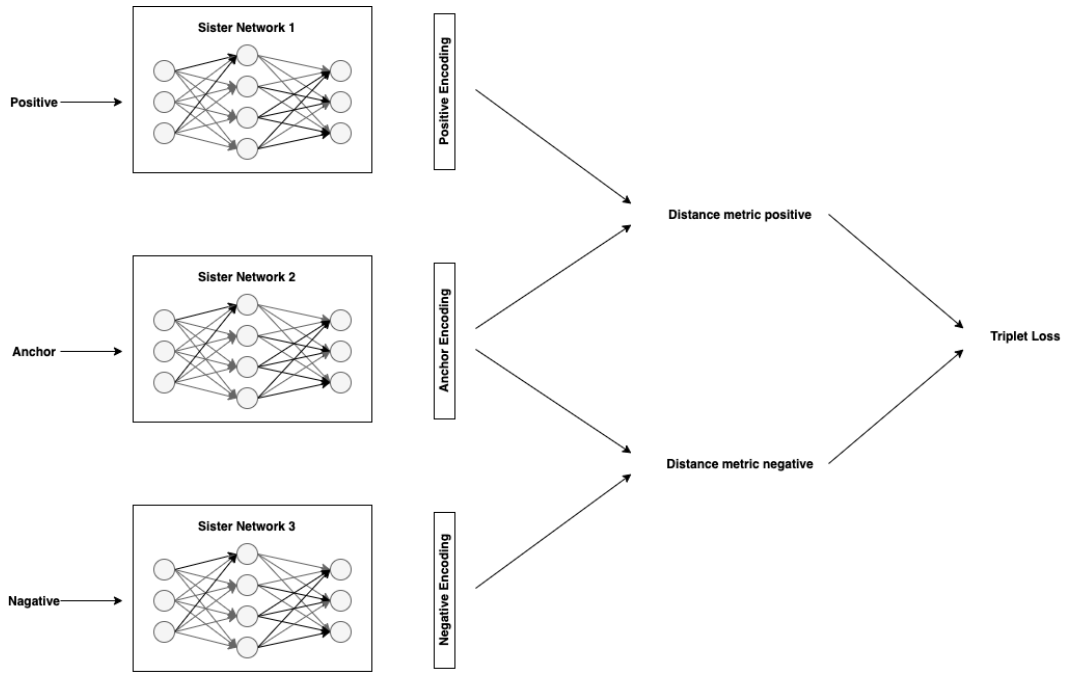

Figure 2.20: SNN with three input branches and a triplet loss function.

Just as for the bi-branch architecture with BCE loss, the goal of the triplet loss function is to minimize the vector distance between pairs from the same class, and maximize the vector distance between pairs from differing classes. However, with triplet loss, the network is fed three distinct inputs; the anchor, the positive, and the negative. The distance between the anchor and the positive input as well as between the anchor and the negative input are computed. If the former distance is smaller than the latter, the model successfully managed to distinguish alike pairs from differing pairs. These two distances can be combined to form the full loss function according to

$$\mathcal{L}\left(x_A, x_P, x_N\right) = \max\left(\mathrm{d}\left(x_A, x_P\right) - \mathrm{d}\left(x_A, x_N\right) + \alpha, 0\right) \tag{2.24}$$

where x_A is the embedding of the anchor input, x_P the embedding of the positive input, x_N the embedding of the negative input, α a margin, and d a pre-determined function evaluating the distance between x_0 and x_1 in vector space. Using the euclidean distance as function d yields the loss function

$$\mathcal{L}\left(x_A, x_P, x_N\right) = \max\left(\| \left(x_A - x_P\right) \|^2 - \| \left(x_A - x_N\right) \|^2 + \alpha, 0\right) \tag{2.25}$$

An additional loss function which has proven useful when using SNNs is contrastive loss. Contrastive loss utilises the same bi-branch architecture as seen in Figure 2.19. The loss function is defined as

$$\mathcal{L}\left(y, x_0, x_1\right) = y \cdot \mathrm{d}\left(x_0, x_1\right) + (1 - y) \cdot \max\left(0, \alpha - \mathrm{d}\left(x_0, x_1\right)\right) \tag{2.26}$$

where y is the target label (0 or 1 depending on if pairs are from same class or not), x_0 and x_1 the embeddings produced for the input pairs, α the margin, and d is the distance metric. Using the euclidean distance as function d yields the loss function

$$\mathcal{L}\left(y, x_0, x_1\right) = y \cdot \|x_0 - x_1\|^2 + (1 - y) \cdot \max\left(0, \alpha - \|x_0 - x_1\|^2\right) \tag{2.27}$$

When feeding the network unbalanced inputs, it can be useful to be able to alter the weights which each input class carries with regard to the loss function. This is done in order to remedy any potential bias towards over-represented classes in the dataset. Weighted binary cross entropy (WBCE) is a form of weighted loss function. WBCE is identical to regular BCE, but with class specific weights. The WBCE loss function can be defined as

$$WBCE(y, \hat{y}, w_1, w_2) = -\frac{1}{N}\sum_{i=1}^{N}(w_1 \cdot y_i log(\hat{y}_i) + w_2 \cdot (1 - y_i) log(1 - \hat{y}_i)) \tag{2.28}$$

with y and $\hat{y}$ being the ground truth and the predicted outputs, and w_1 and w_2 being the weights of the classes, respectively. The weight for each input can be calculated as the proportion of instances from a certain input class in relation to the size of the whole dataset.

2.9 Evaluation Metrics

The performance of a classifier can be evaluated using its prediction scores across all classes contained in the dataset. The test set is fed to the network in order to obtain predictions for every instance. Comparing these predictions with their respective ground truth labels, the number of correct and incorrect classifications made by the network can be calculated. A confusion matrix; a visualisation of the predictive abilities and inabilities of the model, can be constructed using these results. The confusion matrix displays the number (or proportion) of correct, and incorrect, classifications made across every class of the test set. This evaluation method therefore provides an indication of the model performance with regard to each individual class. The goal is to obtain 100% correct classifications and 0% incorrect classifications for every class.

The structure of a confusion matrix in the binary classification case can be seen in Figure 2.21.

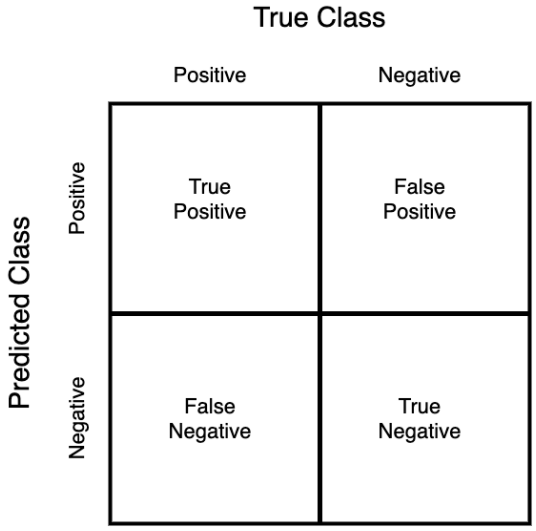

Figure 2.21: Confusion matrix for a binary classifier.

SNNs are a form of binary classifiers, however they operate slightly differently to regular ANN classifiers. Using the embeddings produces by the SNN, we can calculate pairwise distances used to distinguish between classes. The classification of pairs therefore requires a similarity threshold to be established. Pairwise distances between embeddings over that pre-determined threshold implies a classification of the pair as originating from differing classes, and pairwise distances beneath that threshold implies a classification of the pair as originating from the same class. The threshold has to be chosen carefully in order to maximise the predictive abilities of the network. If the value of the threshold is chosen to be excessively high, the result will be a model which predicts most of the input-pairs as coming from the same class, leading to many true positives (TP), but also many false positive (FP) predictions. On the contrary, if the threshold is too low, the model will predict too many false negatives (FN), but catch most true negatives (TN) as well. This is the issue of sensitivity versus specificity. Sensitivity, or true positive rate, can be defined as $\frac{TP}{TP+FN}$ while specificity, or true negative rate, can be defined as $\frac{TN}{TN+FP}$. This delicate balance between sensitivity and specificity is one which needs to be navigated carefully. The threshold should be chosen appropriately depending on the application of the SNN.

The sensitivity and specificity of a model can be plotted for differing threshold values. This plot is called the Receiver Operating Characteristic (ROC) curve. An example ROC curve can be seen in Figure 2.22. The dashed line in the Figure depicts the classification abilities of a randomly guessing classifier. The continuous black curve illustrates the ROC curve for an examplified classifier. Movements along the curve, caused by a change in the threshold, give rise to changes in both sensitivity and specificity of the classifier. At either end of the threshold spectrum we see that the sensitivity and 1-specificity become zero and one.

Another important metric which can be used to evaluate the model is the Area Under the ROC curve (AUC). This is a measure of the classification abilities of the model with regard to the false positive and true positive rates. This can be computed as the

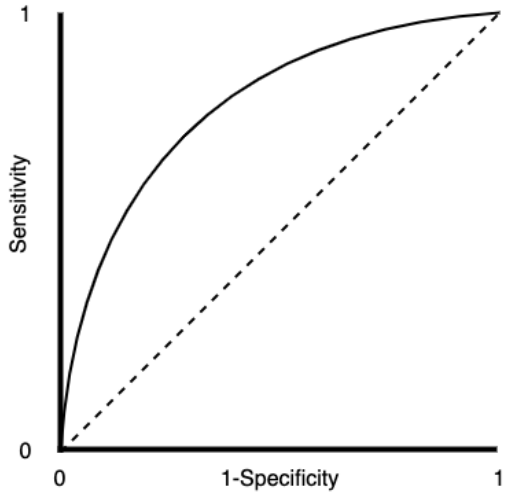

Figure 2.22: Receiver Operating Characteristic Curve.

area beneath the ROC curve. The AUC score for the random classifier (dashed line) therefore equates to 0.5. The goal of a classifier is to maximise the AUC; with the maximum AUC score possible being 1. The perfect AUC score is achieved when the ROC curve extends towards the top left of the plot in Figure 2.22, at which point we obtain a true positive rate of 100% whilst the false positive rate remains at 0%.

Furthermore, the t-SNE algorithm was used throughout the project as a sanity check, in order to provide an indication of how well the models were able to separate the different classes in the training and test sets. This was done through feeding the embedding vectors produced by the network to the t-SNE algorithm and plotting the resulting vectors in 2D space.

3 Method

In this chapter, we provide an overview of the methods used to conduct the research presented in this report. We disclose information regarding the dataset, the processing of the data, the extraction of features from the data, the modelling of the SNNs, and the evaluation of the networks.

One of the sub-goals of this thesis was to investigate if it is possible to use SNNs to detect changes in a person's voice over time. Due to limited availability of data from people with vocal cord cancer, it is unfeasible to train the network to discriminate solely on voice features associated with the disease. In this thesis, another approach is therefore taken. The networks are trained to discriminate on individuals, with the assumption that the voice changes experienced during the progression of vocal cord cancer also leads to an altered voice state. These networks should be able to detect any deviations from the normal state of that individual's voice.

3.1 Dataset

The dataset used in this work is comprised of a series of non-public recordings collected between 2018 and 2021. The recordings were collected through the application VoiceDiagnostic [23] available both on Android (Google play Store) and Apple (App Store) devices. Selected individuals, and patients related to specific clinical studies were invited to the closed platform and asked to record their voices at specific time intervals. This enabled the collection of data from a wide range of subjects whilst obtaining several recording per individual. The ultimate goal of the recording collection is for their usage in constructing mathematical models which can be used to analyze speech patterns in order to offer personalized diagnosis and track recovery of patients after treatments. The work presented in this thesis falls within that scope.

The dataset is comprised of several thousand unique recording sessions made by the users. At every recording session, the user is asked to make two recordings; one in which they hold a sustained "AHH"-sound for as long as they can (named recording-A), and a second recording where they read a short text out loud (named recording-T). The recordings are made on the subjects' phone and then uploaded to a central database containing all recordings. The recordings were sampled at a sampling rate of 44kHz and recorded in stereo (two channels). Furthermore, the following set of datapoints, entered by the user prior to each recording, accompany every recording session:

- Gender of subject (Male/Female/Non-Binary/Unspecified)
- Age of subject (Numerical)
- Cold (True/False)

- Sore throat (True/False)
- Mucus in throat (True/False)

Due to the fact that the subjects were asked to hold the "AHH"-sound for as long as possible during recording-A, the length of these recordings varied greatly. Although to a smaller extent, the same applies to the second recording where the subject often varied his/her speech tempo across recordings. An example of a set of recordings (recording-A and recording-T) from a recording session, for a randomly chosen user, can be seen in Figure 3.1. In this particular case, the recording-T is almost five times as long as the recording-A. However, the recording-T has substantially more silent segments as compared to recording-A. Observing the proportion of silent frames is therefore a means of identifying if a certain recording contains spoken language or single vowels (such as the "AHH"-sound).

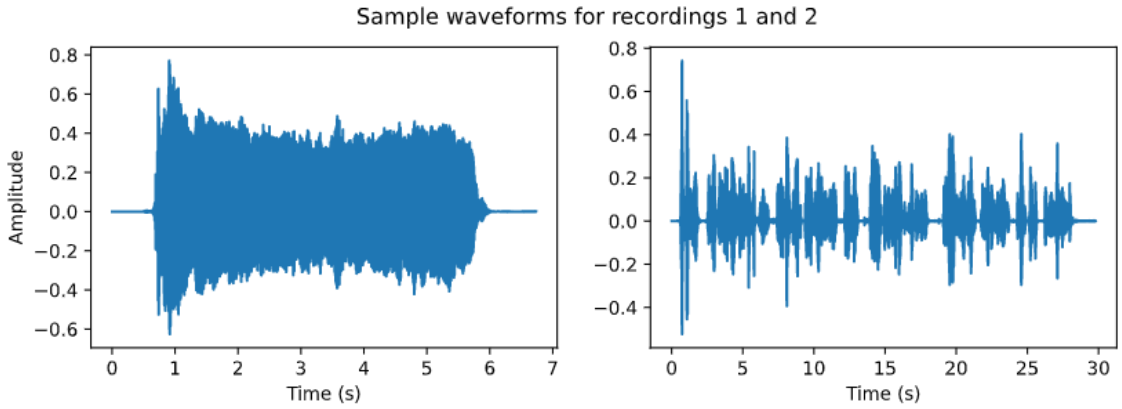

Figure 3.1: Waveforms for sample recording-A and recording-T.

The dataset contains recordings from, among others, users whom have previously been diagnosed with vocal cord cancer. Some of these users have undergone surgery for the disease during the recording period. We therefore have access to recordings both prior to and after surgical intervention. Furthermore, the physician who operated these patients also disclosed the exact date which the relapsing vocal cord cancer was detected. We can test the models on the recordings of these patients during the periods leading up to the surgery. Hopefully, we should be able to see a gradual increase in the distance of the hypothesised voice from the "healthy" state of the voice as displayed in Figure 1.2. These patients' recordings were therefore reserved for testing purposes and not used during the training and validation stages of the model.

3.2 Data Pre-Processing and Feature Extraction

As previously mentioned, the pre-processing of the data and feature extraction was performed in MATLAB. The first step in the pre-processing was to downsample the audio and convert the signal from stereo (two channels) to mono (single channel). These changes reduce the file-sizes substantially, and makes it easier and quicker to

manipulate and store the data. This is beneficial when processing large amounts of data.

Each audio signal was broken down into fixed size sub-segments of amplitude measurements, called frames. Certain frames of the recordings contain information which is not representative of the subjects' voices and should therefore be excluded. An example is the silence at the start and end of some recordings. In order to keep only parts of the recording which contained user specific information, only non-silent frames were retained. The distributions for the length of the recordings (non-silent frames only) can be seen in Figure 3.2. Audio samples from recording-T are on average longer than those from recording-A, however the difference is not as pronounced as the length difference seen in Figure 3.1.

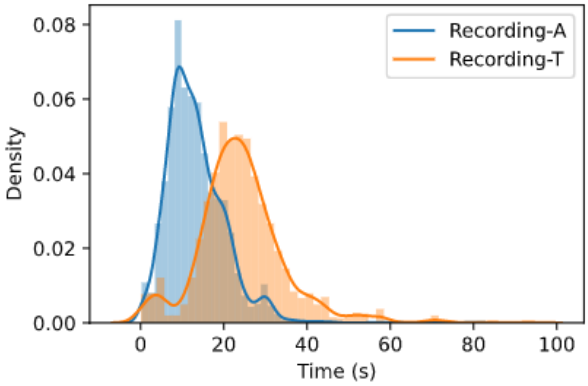

Figure 3.2: Distributions for lengths of recordings (non-silent frames) for recording-As and recording-Ts.

Certain users had to be excluded from the dataset due to irrelevance. For example, participants from some of the studies were excluded as they may have experienced a change in their voices already before making their recordings, or alternatively experienced voice changes change throughout the recording period. This would have hindered the learning process as the networks require non-changing recordings from healthy subjects.

Certain recordings were excluded due to error on the part of the user. Some users had, accidentally or not, avoided following the recording instructions in the application. For example, some users spoke during the recording-A. These recordings were detected through extracting the percentage of pitched frames and discarding the recordings with fewer pitched frames than 80%.

The distribution of the accompanying datapoints for the remaining recordings can be seen in Figure 3.3. Only around 6% of all recordings were performed when the subjects were experiencing flu-symptoms. This number is even lower in the case of sore throat and mucus. We need to account for this fact when building models which are able to distinguish between "normal" voice changes (benign voice changes) such as the one experienced when suffering from a cold, and malignant voice changes associated with diseases such as vocal cord cancer. This is explored later in the thesis. Furthermore, there are a greater number of men represented in the study as compared to any of the other gender categories. This may have an impact on the final performance of the

model, as it may lead to bias towards vocal characteristics and features more often observed in men.

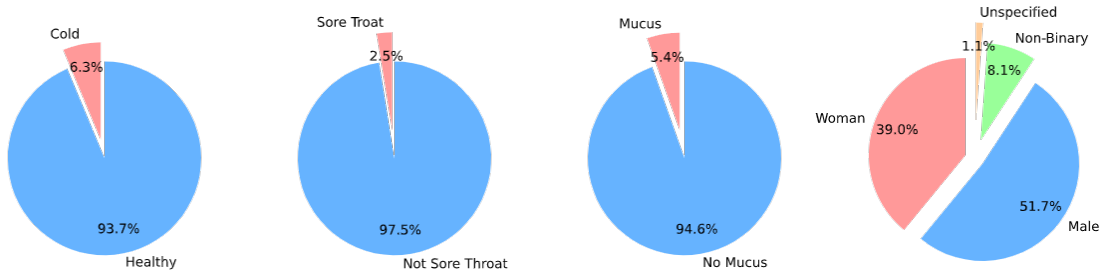

Figure 3.3: Distributions for accompanying datapoints.

Audio samples from recording-A are monotonous and less phonetically varied than those from recording-T. Samples from recording-T contain both pronounced vowels and consonants whilst those from recording-A only contain the utterance of a single sound and vowel; namely the letter "A". This makes recording-T more complex in comparison and potentially more information-dense regarding the state of the subjects' voice. Due to the relative simplicity of samples from recording-A, these were initially used in the training of the models, with the goal of modelling with the recording-T at a later stage.

After the pitched frames had been extracted, the STFT was applied to these frames. Taking the magnitude squared of the transformed frames yielded the spectrogram of every recording. An example of the visualised spectrograms for 6 randomly chosen sample recording-As can be seen in Figure 3.4. The different frequencies and their respective intensities remain relatively constant for every recording. This indicates that the frame features do not differ significantly throughout separate recording instances. Therefore, the frame features should remain relatively constant during every recording.

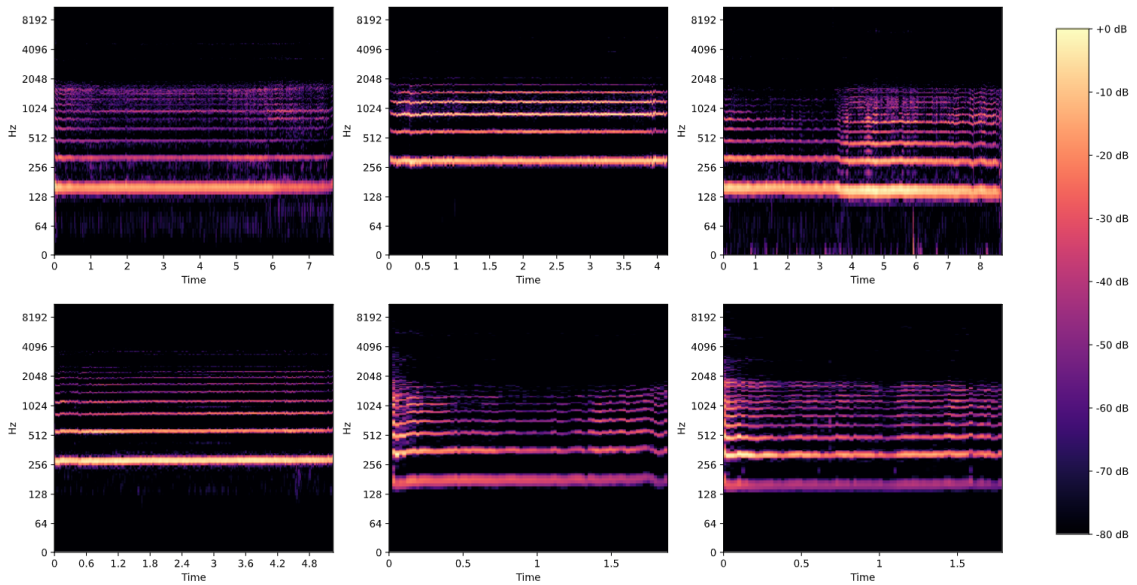

Figure 3.4: Spectrograms for 6 randomly chosen sample recording-As.

A mel-filterbank with various filters was applied to the spectrogram in order to obtain

the mel-spectrogram. The mel-spectrogram of the same 6 sample recording-As from Figure 3.4, can be seen in Figure 3.5. The spectrogram in Figure 2.3 was divided into 1024 frequency bands where as the mel-spectrogram has fewer, but larger, mixed-size bands covering the various parts of the frequency spectrum.

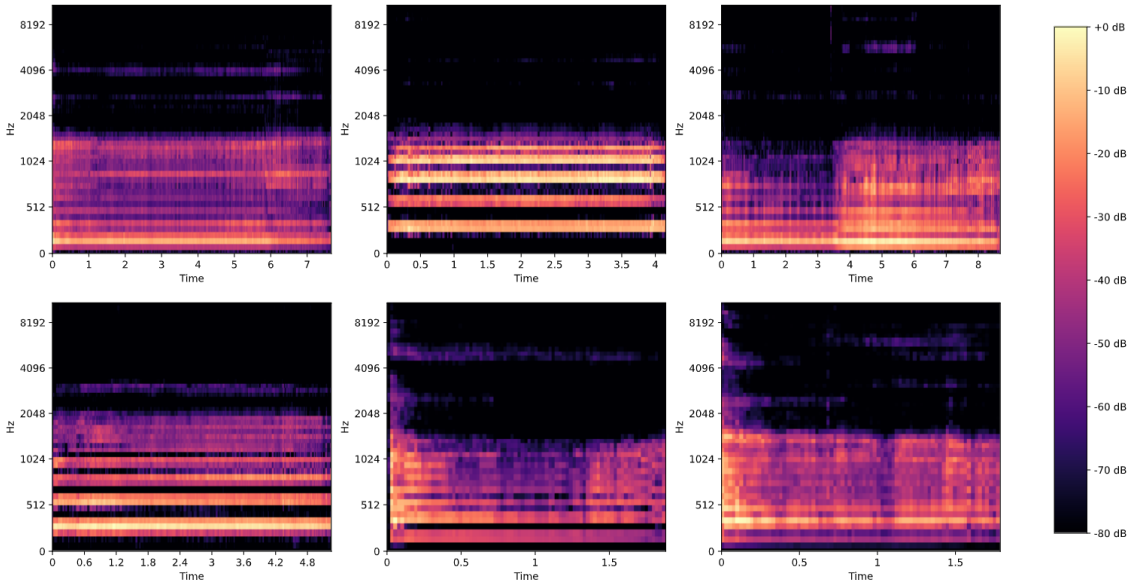

Figure 3.5: Mel spectrograms for 6 randomly chosen sample recording-As.

Next, the MFCCs were computed trough applying a discrete cosine transform (DCT) to the frames of the mel-spectrogram. The DCT de-correlates the features and retains most of the information in first few coefficients. The MFCCs of the same 6 sample recording-As from Figure 3.4, can be seen in Figure 3.6.

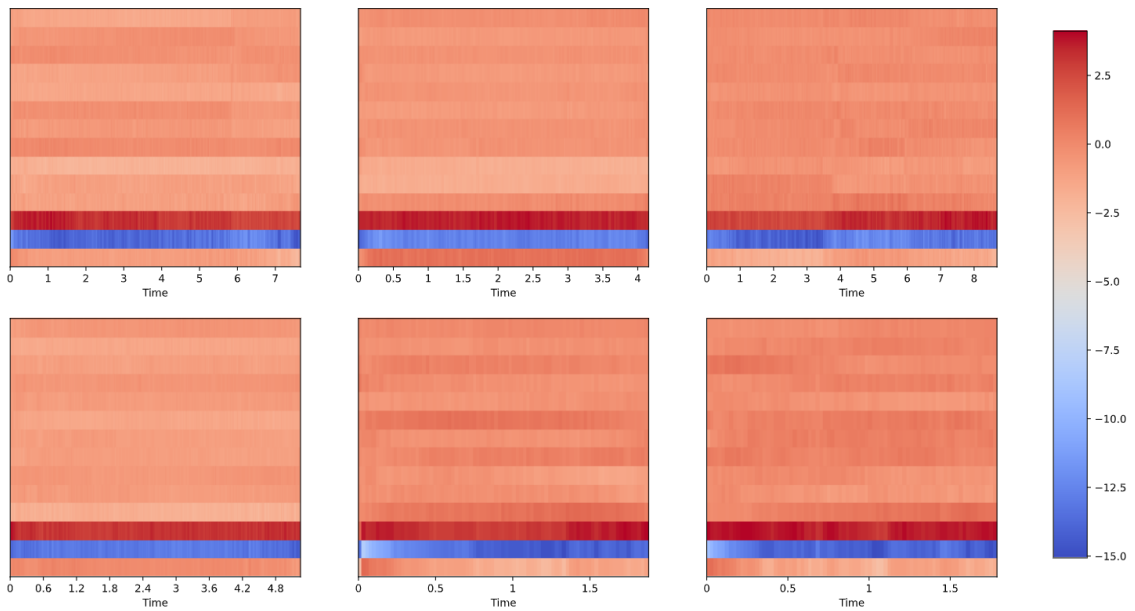

Figure 3.6: MFCCs for 6 randomly chosen sample recording-As

Looking at the spectrogram, the mel spectrogram, and the MFCCs for the different recordings, we can observe that none of the features vary significantly with respect

to time. This is evidently due to the aforementioned phonetically monotonic nature of recording-A. The question that arises is whether the network should be trained on features for whole recordings (longer sequences of frames), or on features for separate frames. The benefit of passing features from whole recordings to the network is that the network can learn global features from the recordings as opposed to local features on frame level. However, since the features do not seem to vary a significant amount over time, it may be beneficial to choose the frame-level perspective as it provides the network with more data samples to train on. In this thesis, we attempted training the networks with data on both frame and recording level.

To eliminate further bias and overfitting from the models, the dataset was split into three subsets; the training set, the validation set, and the test set. The test set was comprised of the recordings of the patients whom we suspected had experienced a voice change during the recording period (those with a relapse of their vocal cord cancer for example), and the rest of the recordings were allocated to the train and validation set. The train-validation split was performed with the ratio 80:20 in favour of the training set. Furthermore, the split was performed on user level, not recording level. This implies that all recordings from 80% of the users were allocated to the training set, while the other 20% of users' recordings went to the test set. This ensured that the models would not be biased towards the specific users of this dataset, and that they could generalise well to never before seen users. The same users were chosen for the training and validation sets across every model. This enabled a more fair comparison of model performance as all models were trained and tested on the same data. This was achieved through initialising the pseudorandom number generator used to split the dataset with a set seed.

The training set was standardised to Gaussian distributions by removing the means and scaling to unit variance. These transformations were applied to the validation and test sets using the same standardiser.

3.3 Modelling

Several model architectures were evaluated in order to obtain optimal performance with regard to the pre-determined evaluation metrics. Initially, models were constructed and trained on frame-level inputs. This approach provided more training data since each recordings could often be split into several hundred different individual frames. This meant that these models could be trained for a much longer duration. The frame level approach provides more granular output surrounding the state of a recording. The input to the frame level models is uni-dimensional and the length of the input is determined by the type of feature selected. For example, using the spectrogram meant an input vector with the size of 1024 for every frame. For the mel spectrogram and the MFCCs, that number was significantly smaller. Recording level models were constructed in order to compare performance. These models output only one similarity measure per recording, and thus, are easier to interpret.

The basic skeleton of the SNN can be used with many different types of sub-network architectures. The structure of the base-network can be altered to fit the type of input

required. This changes the dimensions of the left and right input branches. When using triplet loss as a loss function, the network structure needs to be altered to an architecture with three input branches.

With one-dimensional (1D) inputs, regular ANNs with dense layers can be utilised as sub-networks. Alternatively, networks with 1D convolutional layers can be used. The kernel size and the number of output filters in the convolutional layer of the 1D convolutional network was chosen through trial and error.

The health state associated with each recording was also utilised in the modelling. Information on whether the patient was experiencing symptoms of a cold, sore throat, or mucus in the throat, were all provided by the users themselves at every recording instance. Extending the frame-level models above, a model was constructed to output not only the similarity score between a pair of inputs, but also a second output stating the probability of an altered non-malignant voice change (cold for example). The left output branch of this model was the cold classifier, and the right branch was the Siamese distance metric.

Since we only want the model to recognize benign illness (cold, sore throat, etc) if the input pairs come from the same user, only same class pairs have the potential to contain a "sick recording". The model is trained with healthy recording features fed to the left branch and a mix of healthy and sick features to the right branch. This maintains a consistency in the inflow of input pairs in order for the network to learn in an ordered manner. With only same-class pairs being able to contain features from benign illnesses, the overall batch probability of a pair containing such a recording decreases to 25%. This imbalance in the inputs can cause severe bias in the network and hinder learning. For this reason, when using the health data, the WBCE loss function was utilised.

The limitation of frame-level models is that they loose the macro perspective. This can be remedied through shifting to models which are fed features for a whole recording. The sub-networks can be altered to 2D convolutional networks. In this case, a relatively simple convolutional architecture was chosen as the sub-networks. Additionally, pooling layers were added after the convolutional layers. Lastly, a flattening layer was introduces after the last max pooling layer in order to transform the 2D output into a 1D vector which could be fed to a final dense layer.

Keras provides several pre-conFigured loss functions. Among those is the BCE, which is compatible with the majority of the previous models. When switching to contrastive loss or triplet loss, a custom loss function needs to be constructed as a function within Python, and provided to the model compiler. The compiler makes calls to this function when calculating the loss after each iteration of the network. Constructing a custom loss function is relatively simple. The function takes the predicted output and the ground truth output as required inputs, and return the calculated loss for those given values. In order to process batches, the method should be capable of accepting an array of inputs and return an array of losses as outputs. The triplet loss function accepts two inputs, the Anchor-Positive and the Anchor-Negative distances, and returns a single loss as output.

The tri-branch architecture utilised a custom triplet loss function. The triplet model does not produce an output in the same way as the bi-branch architecture does. The pairwise distances between inputs have to be calculated manually through extraction of the sub-networks. The loss is calculated through a combination of the Anchor-Positive and Anchor-Negative pairwise distances between embeddings. The base model can yet again conform to any sub-network architecture desired.

The recording level model was also extended to accept two input-branches simultaneously. The first branch being the same features previously mentioned (left branch), and the second being 67 different recording level features (right branch) explained in the chapter on theoretical background. The left branch contained 2D inputs and required 2D convolutional layers, where as the right branch contained 1D inputs and required dense or 1D convolutional layers. In this case, dense layers were chosen for the right branch.

The distance measure used for all network architectures was the Euclidean distance. Both Manhattan distance and cosine similarity were tested, however the Euclidean measure yielded superior results. When using the Cosine distance, a minimised loss lead to models which were no better than randomly guessing classifiers.

All the networks were trained using batches of input pairs and the mini-batch gradient descent algorithm. Batches were sent to the networks with half of the pairs containing recording features from alike users and the other half from differing users. A visualisation of this can be seen in Figure 3.7. In this case, full coverage of the dataset implies pairing every recording with every other recording in the dataset. This would have resulted in almost 1,000,000 different input pairs (in the frame case, it would have been even more) which is unfeasible. For this reason, the batch generator samples the classes, and recordings therein, randomly. This makes it difficult to ensure every recording is fed to the model in every epoch, but over many epochs we can with relative certainty ensure a good coverage of the dataset. The number of steps per epoch is normally set to the length of the dataset; in the case of most models in this thesis, it was set to a value of 2000.

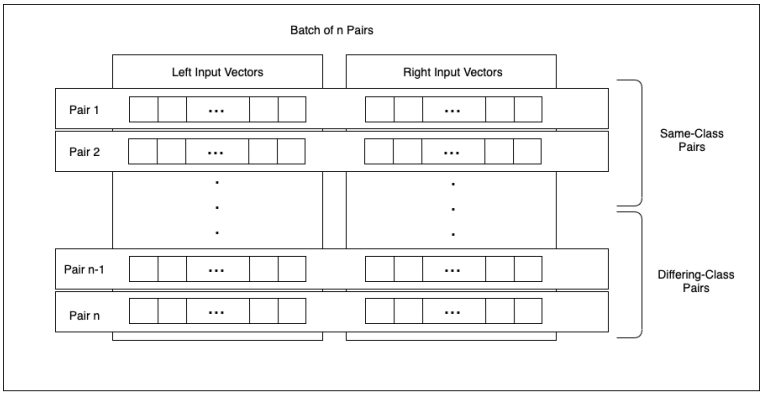

Figure 3.7: Batch of n input pairs.

The smaller we can make the latent space, the fewer parameters will be required in the overall network architecture. This also means that the embeddings will take up less space if they are to be stored and used at a later date. If the dimensions of the latent space are made too small, the network will not be able to represent the

features within the embeddings and this will hinder the network from learning. As the latent space increases in size, there will come a point when the increase will no longer impact the feature extracting abilities of the network. Any increase after this was found to lead to a zero-padding effect of the embedding vector. In order to find the dimensions of the latent space which is sufficiently small whilst not being limiting for performance, different sizes were examined and evaluated with respect to the AUC score on the validation set. The optimal embedding size varied somewhat between model architectures and was therefore chosen according to the circumstances at hand. An example of the embedding size plotted against the AUC score for one of the models investigated can be seen in Figure 3.8.

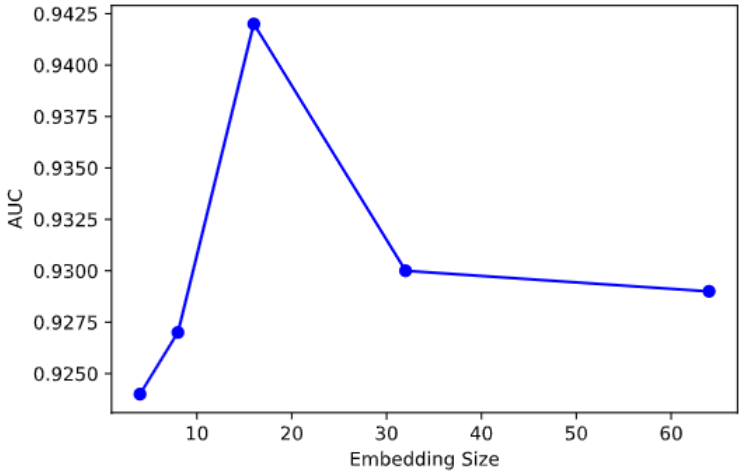

Figure 3.8: AUC score for models with different embedding sizes.

3.4 Evaluation

In order to evaluate the network architectures, the sub-networks were extracted and used to make predictions on the validation set. Even with differing macro structures, such as the triplet network, the task of the sub-network remains identical across models; namely to calculate the embeddings. Once the embeddings are calculated, the distances to other recordings in vector space are easily computable using any of the distance measures from the previous chapter. The Euclidean distance measure was chosen for all models. Distances between the same class recordings should be beneath a given threshold and distances between differing class recordings should be above it. Through variations of the threshold, the TP, FP, TN, and FN rates can be computed and used to create the ROC curve and associated AUC score. Figure 3.9 shows an example of the ROC curves of an untrained model and a trained model. Note that the AUC of the untrained model is not 0.5. This is because the randomly initialised weights in the network created an embedding space which was capable of separating classes better than a random guessing classifier.

The non-transformed input feature vectors can be used to create a baseline classifier. The distance between these feature vectors were calculated identically to the embeddings, and used to create a ROC curve for each feature. The ROC curves for the

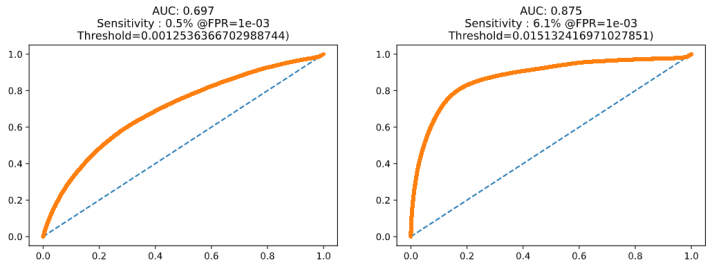

Figure 3.9: Example ROC curves for untrained (left) and trained (right) networks.

baseline classifier can be seen in Figure 3.10. Interestingly enough, the AUC score for the baseline classifier using the MFCCs is very high, and thus the question arises if SNNs are even needed in order to distinguish between voices.

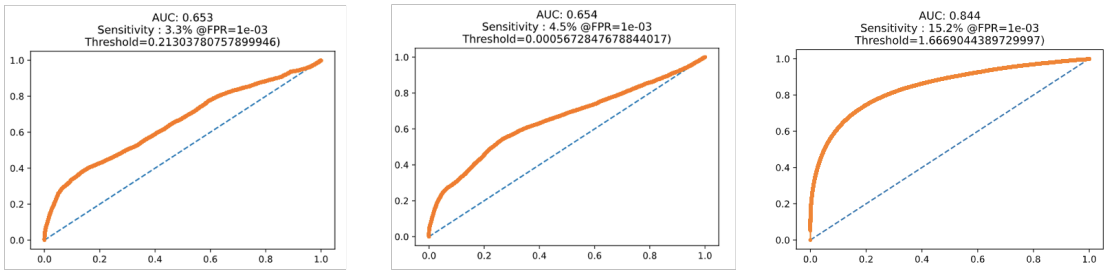

Figure 3.10: Example ROC curves for baseline classifiers using the spectrogram (left), mel spectrogram (middle), and MFCCs (right) as embeddings.

A second approach to evaluating the models is to visualize how well they are able to separate recordings in vector space. This can be done by applying the t-SNE algorithm to the embedding vectors for every recording or frame. An example of a t-SNE visualisation of the embedding vectors, for both an untrained and a trained model, can be seen in Figure 3.11. As seen with the ROC curve, the untrained model is able to separate the different users' recordings relatively well. However, the clustering is tighter for the trained model, and the inter-cluster distance is larger between users.

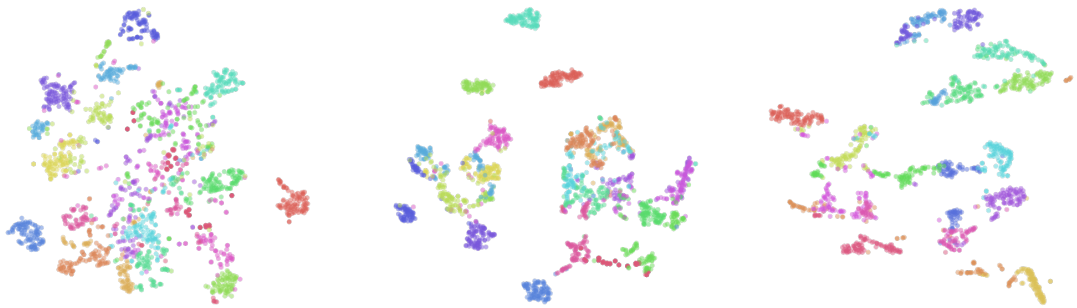

Figure 3.11: t-SNE visualisations of embeddings for an untrained model (first image) and a trained model for training set (second image) and validation set (third image).

Lastly, the models can be evaluated by looking at the distributions of the distances between embeddings. Embedding pairs for same-class inputs should be distributed around zero as these embeddings should lie close to each other in vector space. Pairs from differing classes should be centered around a positive and non-zero value. This value depends on the network architecture. If the network output is bound between zero and one, the maximum value is one. If the network output is unbounded however, there is no maximum value which the distance can take. Since the distance cannot be smaller than zero the distribution for alike pairs can be expected to be truncated at zero. The probability density function (PDF) of the distances can provide an indication of the classifying abilities of the model and on what of the two categories the model is performing best. Furthermore, this can be made even more granular through breaking the distributions down into the respective users. This enables a more detailed analysis of the model performance with regard to different classes. An example of what the PDFs could look like for a model with good performance can be seen illustrated in Figure 3.12. The more overlap between the two PDFs, the harder it is for the model to distinguish between alike and differing pairs.

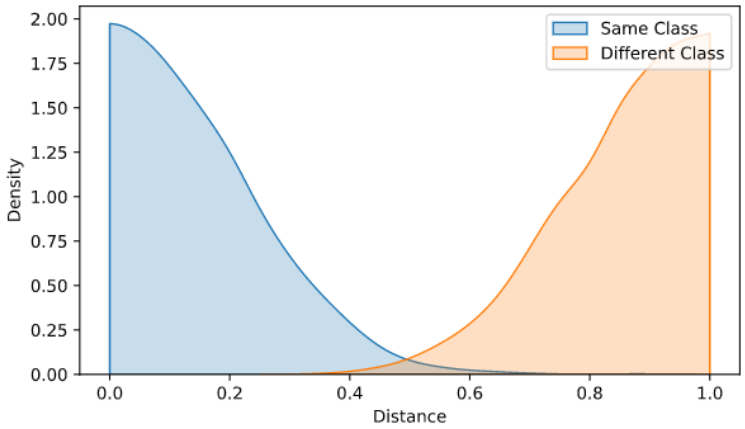

Figure 3.12: Probability density functions for alike pairs (blue) and differing pairs (orange), for simulated data.

3.5 Testing

There were five users in the dataset with recordings, whom we knew had had a recurrence of their vocal cord cancer. Only one of those five users however had performed recordings prior to the remission of the cancer. The recordings prior to this patient's diagnosis were used in order to verify if there was any noticeable alteration of the patient's voice during this time period. Furthermore, one of the patients had undergone surgery a month before he/she had started to make recording on the application. The recordings following the surgery were also included in the test set in order to verify if there were any detectable voice changes during the recovery period.

Due to the relatively few test-subjects contained in the dataset, the recordings of another study were utilised to test the models. These patients were also part of the original dataset, but had been excluded at an earlier stage due to the fact that their voices had most likely changed throughout the study. These were patients who had

been diagnosed with gender dysphoria and were undergoing treatment and coaching to alter their voices. Their recordings were used to verify of the models could detect these voice changes.

In total, 4 separate patients were identified as test subjects and who's recordings were used to test the models. Included in the test set were the following patients:

1. **Test patient 1:** Patient who's vocal cord cancer relapsed. Recordings begin around 10 months prior to diagnosis of the cancer.
2. **Test patient 2:** Patient who had undergone surgery 2 months prior to the beginning of recordings
3. **Test patient 3:** Patient who was trying to alter the state of their voice due to gender dysphoria
4. **Test patient 4:** Patient who was trying to alter the state of their voice due to gender dysphoria

All of the previously mentioned audio features were extracted from the recordings made by these four users. These features were then fed to the base networks in order to create embeddings for every frame, or recording. In order to compare frames or recordings, an embedding "anchor" representing their voice's base state was created for every user. The anchors were computed as the mean of the embeddings created from the user's first recording during the investigated period. The distance between the anchor embedding and the embedding from the other frames or recordings could then be computed using the euclidean distance measure. One frame level model, and one recording level model was chosen to evaluate the potential voice changes of the four test subjects. The frame level model chosen was the SNN_CL_R, and the recording level model chosen was the SNN_CL_R.

4 Results and Discussion

Throughout this chapter, the results of the work is presented, and a discussion is conducted surrounding the implications of these results. The different model architectures are evaluated, on both frame and recording level, in order to find the model with the best performance. First, the frame level models are investigated, followed by the recording level models. Furthermore, all model variants were tested on differing audio features in order to establish best practices regarding optimal feature extraction and selection. For every model architecture, the AUC score is reported along with the optimal input features. The models are compared to the baseline classifiers which calculates the distance between the standard feature vectors/matrices and thereafter calculates the AUC score for these.

4.1 Frame Level Models

The frame level models were trained on input pairs of features from individual frames as opposed to features from a sequence of frames stacked together. The inputs to these models were batches of 1D feature vector pairs, together with the binary labels relaying whether the features were picked from the same class or not. During the evaluation of the models, it was established that 1D convolutional layers achieved superior results to the network structures containing dense layers. For this reason, 1D convolutional layers were chosen for all final models presented below. The following three models, plus one baseline classifier, were trained and tested:

1. **BL_F:** Baseline model using frame level recording features.
2. **SNN_BCE_F:** Siamese Neural Network with 1D convolutional layers. Binary Cross Entropy loss function. Trained on frame level features.
3. **SNN_CL_F:** Siamese Neural Network with 1D convolutional layers. Contrastive loss function. Trained on frame level features.
4. **SNN_TL_F:** Siamese Neural Network with 1D convolutional layers. Triplet loss function. Trained on recording level features.

The frame level model training times and number of epochs depended on the features used and the architectures of the model. On average, the training duration was around 5 minutes and every model was trained until there was no longer any significant decrease in the training loss, or until the validation loss started to increase. In order to reduce bias, 100 frames were chosen, at random, from every user for use during training.

Similar hyperparameters were used to train all three models, with only slight variations in embedding size and regularisation amount. The hyperparameters for the respective models can be seen in Table 4.1. These were chosen according to trial and error, and could therefore be further tuned to optimise the performance. The term α in the Table refers to the margin used in the contrastive loss and triplet loss functions.

HP: SNN_BCE_F	
Learning rate	0.0001
Regularization	0.001
Batch size	32
Steps per epoch	1000
Epochs	20
Embedding size	16

HP: SNN_CL_F	
Learning rate	0.0001
Regularization	0.0005
α	1
Batch size	32
Steps per epoch	200
Epochs	50
Embedding size	16

HP: SNN_TL_F	
Learning rate	0.0001
Regularization	0.0001
α	0.2
Batch size	32
Steps per epoch	1000
Epochs	15
Embedding size	32

Table 4.1: Hyperparameters used during the training of frame level models: SNN_BCE_F (left), SNN_CL_F (middle), SNN_TL_F (right).

After training the models, they were all tested on the validation set in order to produce ROC curves and AUC scores. The true positive rates and false positive rates were calculated through classifying 10,000 different input pairs taken from the validation set together with their respective binary labels. Of these 10,000 pairs, 5,000 were same-class pairs, and the other 5,000 were differing-class pairs. This was done in order to maintain balance among the inputs. The different models were trained and tested on the same data. The AUC scores for the different models can be seen in Table 4.2.

Model	AUC Score Train	AUC Score Validation
BL_F	0.918	0.917
SNN_BCE_F	0.941	0.943
SNN_CL_F	0.951	0.950
SNN_TL_F	0.978	0.947

Table 4.2: AUC scores for frame level models and baseline classifier for training set and validation set.

All three SNNs produced similar AUC scores. It is worth mentioning that the AUC scores varied somewhat depending on what users were included in the training set. This indicates there may be users who are more difficult to distinguish than others. The frame level model which performed best was the bi-branch architecture with contrastive loss. Both the models implementing alternate loss functions (triplet loss and contrastive loss) performed better than the model with a simple BCE loss function, validating that these loss functions are an effective way of boosting the performance of SNNs even in the case of audio data. Even though these two loss functions increased the AUC, the difference was minor as seen from the ROC curves in Figure 4.1. The AUC score on the training set for the SNN_TL_F model was significantly higher than on the validation set. This is the only case where we see the model overfitting on the training data. This may have caused a deterioration in the performance on new data overall.

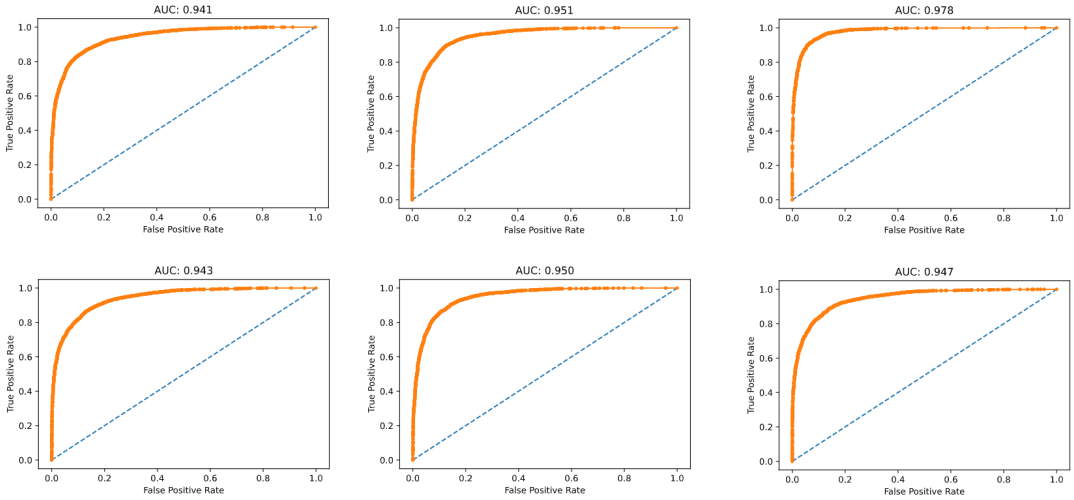

Figure 4.1: ROC curves derived from the training set (top row) and validation set (bottom row) for frame level models SNN_BCE_F (left column), SNN_CL_F (center column), and SNN_TL_F (right column).

The models with the highest performance were those which used the mel spectrogram, MFCCs, pitch, smoothed pitch, HNR, and AR-coefficients as inputs. In all cases, the optimal approach was to concatenate the aforementioned features into a single feature array, and feed these to the model in batches. Using single features as inputs proved less effective than combining them. For example, using the spectrogram alone as input led to significant fluctuations of both loss and accuracy and never led to consistent results or adequate accuracy.

The baseline model with the highest AUC score used only the MFCCs as inputs. All other feature combinations obtained AUC scores in the 0.65 to 0.75 range. Even though the baseline classifier produces an inferior AUC score, it is not significantly lower. This reinforces the representative powers of MFCCs.

As seen in Figure 4.2, all three models are able to form clusters for the different users of the validation set with relative ease. There are certain users whose recordings are more difficult to classify than others, and certain users whose recordings have similar distinguishing features causing them to be relatively close in the vector space. We can also notice that in certain cases there are several sub-clusters for the same user. Frames within these clusters most likely originate from differing recordings where the user may have experienced a slight voice alteration between recording instances.

The distributions of the pairwise distances seen in Figure 4.3, show two very similar distributions for the SNN_CL_F and SNN_TL_F, and a slightly differing distribution for SNN_BCE_F. This can be explained by the additional dense layer with sigmoid activation function added at the end of the SNN_BCE_F, which re-scaled the output between zero and one. The difference between the two models with custom loss functions lies in the magnitude of the distances. The distances are only restricted to be non-negative, hence they are not capped at any maximum value. This implies that differing models may learn embedding spaces producing vectors with vastly differing inter-vector distances. This can be observed in Figure 4.3, also seen in Figure A.1

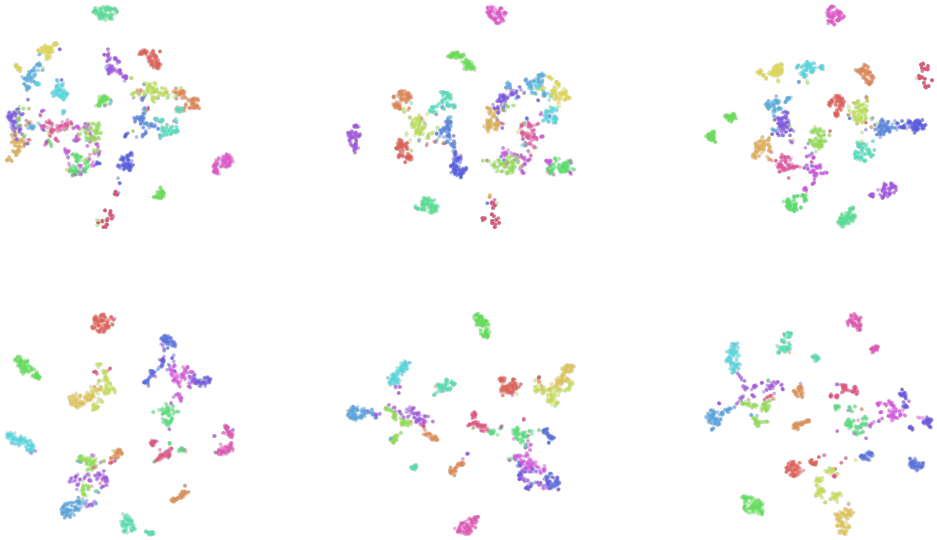

Figure 4.2: t-SNE visualisation of embeddings derived from the training set (top row) and validation set (bottom row) for frame level models SNN_BCE_F (left column), SNN_CL_F (center column), and SNN_TL_F (right column).

through A.3 in the appendix. SNN_CL_F produces embeddings which are closer to each other in vector space, and SNN_TL_F embeddings which are furthest away from each other. The standard deviation of the same-class pairwise distances is significantly lower than that of the differing-class pairs. This is due to the larger range of possible pairwise distances between differing recordings. The similarity of voices from different users is a spectrum whereas in the case of same-class recordings that similarity is limited to a smaller range due to the recordings originating from the same person.

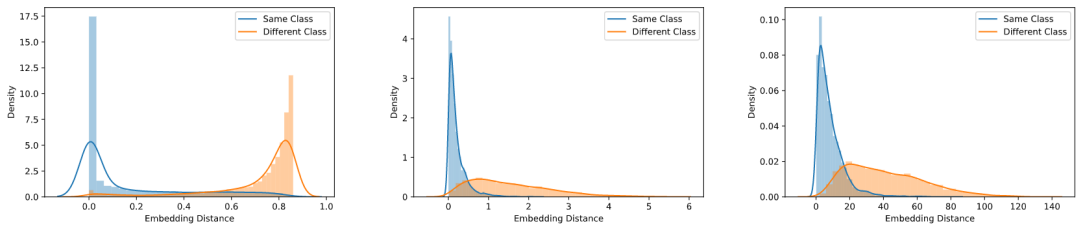

Figure 4.3: Distributions for pairwise same and differing class distances for frame level models SNN_BCE_F (left), SNN_CL_F (center), and SNN_TL_F (right).

The SNN_BCE_F model was also extended with a secondary output branch as shown in Figure **??**. This model will be referred to as SNN_COLD henceforth. For the SNN_COLD model, data was split identically to the previous frame level models, with the addition of the "cold" parameter added as a second ground truth label. The "cold" parameter was provided by the users themselves in conjunction with every recording instance. In order to combat the unbalanced nature of cold-labels in the dataset, the model was compiled with a weighted binary cross entropy loss function. This forced the network not to over-predict a label as "not cold".

This model architecture was more difficult to tune since there were more parameters

overall and the multi-output structure caused unwanted relationships between the two output branches. Since both output branches relied on the sub-networks in order to make their predictions, a fault in one of them lead to this error being propagated to the output in the other branch and hindered learning. Furthermore, updates to the hyperparameters which led to favorable results for one output branch may even prove counter-useful to the other output branch. The hyperparameters used to train this network can be seen in Figure 4.3.

HP: SNN_COLD	
Learning rate	0.0001
Regularization	0.001
Batch size	32
Steps per epoch	1000
Epochs	40

Table 4.3: Hyperparameters used during the training of the SNN_COLD.

The model combined the individual losses of the two output branches into one global loss, which was used during backpropagation. The issue was that the these two sub-losses did not follow the same trajectories, as they changed with differing rates. This can be seen in Figure 4.4. The validation loss in the right branch started to increase around epoch 15 whereas the validation loss in the left branch kept decreasing during that same time frame. An increasing validation loss is a sign that the model is overfitting. This would indicate that, according to the right output branch, training should be stopped around epoch 15, and according to the left output branch, training could continue past epoch 40. After epoch 15 of training, any further training is harmful to the performance of the right output branch whilst increasing the performance of the left. This is reinforced by Figure 4.5 which displays the training and validation accuracies of the model for the right and left output branches. The accuracy of the right branch started decreasing after the peak around epoch 15.

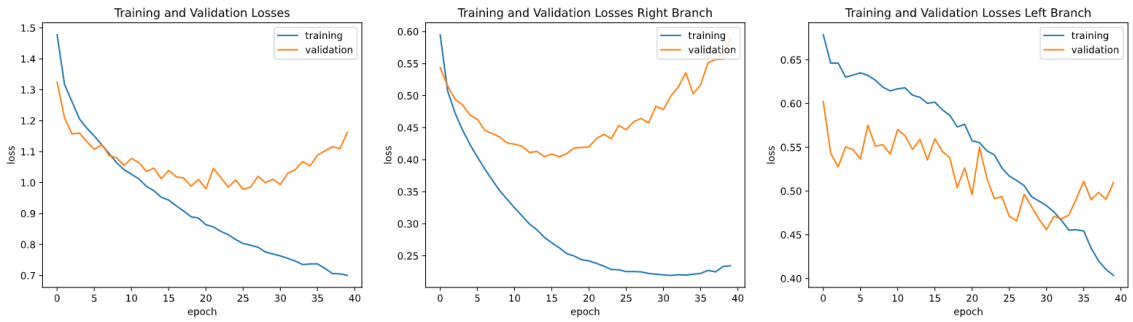

Figure 4.4: Loss during training and validation: overall (left), right branch (middle), and left branch (right).

In the current state, this model does not seem to be a viable option since the increased performance of one output branch comes at the expense of the performance of the other. Furthermore, the data on whether a user had a cold during a certain recording may be unreliable as it was entered by the users themselves leading to highly subjective views and varied data. If there were larger amounts of data and more reliability in the data, this model architecture could prove useful in distinguishing between benign and malignant voice changes.

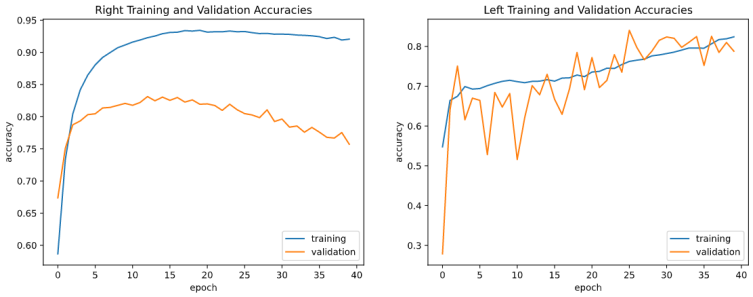

Figure 4.5: Accuracies during training and validation: left branch (left) and right branch (right).

4.2 Recording Level Models

The recording level models were trained on two dimensional features; an array of frame features. These were fed to the network in batches, just as with previous models. The sub-networks were modified with convolutional layers to be able to process these batches of 2D data. After the convolutional layers, the data was flattened into a 1D array which was fed through two dense layers which in turn outputted the feature embedding. The models remained identical to the frame level models on a macro level, and with regard to the loss functions utilised. The following recording level models were trained and tested:

1. **BL_R:** Baseline model using recording level features.
2. **SNN_BCE_R:** Siamese Neural Network with 2D convolutional layers. Binary Cross Entropy loss function. Trained on recording level features.
3. **SNN_CL_R:** Siamese Neural Network with 2D convolutional layers. Contrastive loss function. Trained on recording level features.
4. **SNN_TL_R:** Siamese Neural Network with 2D convolutional layers. Triplet loss function. Trained on recording level features.

The recording level models were trained until the validation loss started increasing. For the SNN_BCE_R it took only a few epochs, with 100 steps per epoch, to train the network. Any further training of the model often lead to a deterioration in performance on the validation set most likely due to overfitting. The models with alternative loss functions required more time to train and could be trained for more epochs prior to overfitting.

The hyperparameters used to train the recording level networks can be seen in Table 4.4 with α again being the margin used in the contrastive loss and triplet loss functions.

Convolutional layers require a fixed size input which needs to be specified during the construction of the networks. In this case, the dimensions of the input data equate to the shape of the feature by the number of frames. The number of frames per recording was chosen to be 50, as this was long enough to gain sufficient coverage

HP: SNN_BCE_R	
Learning rate	0.001
Regularization	0.0001
Batch size	64
Steps per epoch	100
Epochs	5
Filters	32, 64
Filter size	3x3
Embedding size	32

HP: SNN_CL_R	
Learning rate	0.0005
Regularization	0.0001
Batch size	64
Steps per epoch	100
Epochs	20
Filters	32, 64
Filter size	3x3
α	1
Embedding size	32

HP: SNN_TL_R	
Learning rate	0.0005
Regularization	0.0001
Batch size	64
Steps per epoch	100
Epochs	30
Filters	32, 64
Filter size	3x3
α	0.2
Embedding size	32

Table 4.4: Hyperparameters used during the training of recording level models: SNN_BCE_R (left), SNN_CL_R (middle), SNN_TL_R (right).

of the recordings, and short enough for most recordings to be at least this length. The frames were chosen from the middle of the recording in order to ensure that any potential interference, such as silence during the start or end of the recording, were removed. Furthermore, only one frame sequence was extracted from each recording. This was done in order to combat potential bias which would have come from the numerous segments extracted from longer recordings. One segment per recording also meant that there would only be one similarity score for every pair of recordings. This meant less granular output, but also more easily interpretable.

The recording level models were tested on the validation set comprised of the identical users and recordings as the frame level models. The AUC scores for the models, and the baseline classifier, can be seen in Table 4.5.

Model	AUC Score Train	AUC Score Validation
BL_R	0.843	0.842
SNN_BCE_R	0.818	0.816
SNN_CL_R	0.858	0.847
SNN_TL_R	0.923	0.838

Table 4.5: AUC scores for recording level models.

The recording level model with the best performance on the test set was the architecture with a contrastive loss function. Again, the alternate loss function produced superior results to that of the BCE. The ROC curves and their associated AUC scores can be seen in Figure 4.6.

The baseline classifier BL_R achieved a better AUC score on both training and test set over the SNN_BC. The BL_R utilised the MFCCs to determine the similarity between recordings according to equation 2.23.

Contrary to the frame level models, the recording level models performed best when trained on a single feature extracted from the recordings. The features which produced the best results across all model architectures were the MFCCs. The reason why single individual features produced superior results is most likely due to the nature of the convolutional layer. These filters are constructed to be able to find translation-invariant features within the input and are therefore not as effective when separate features

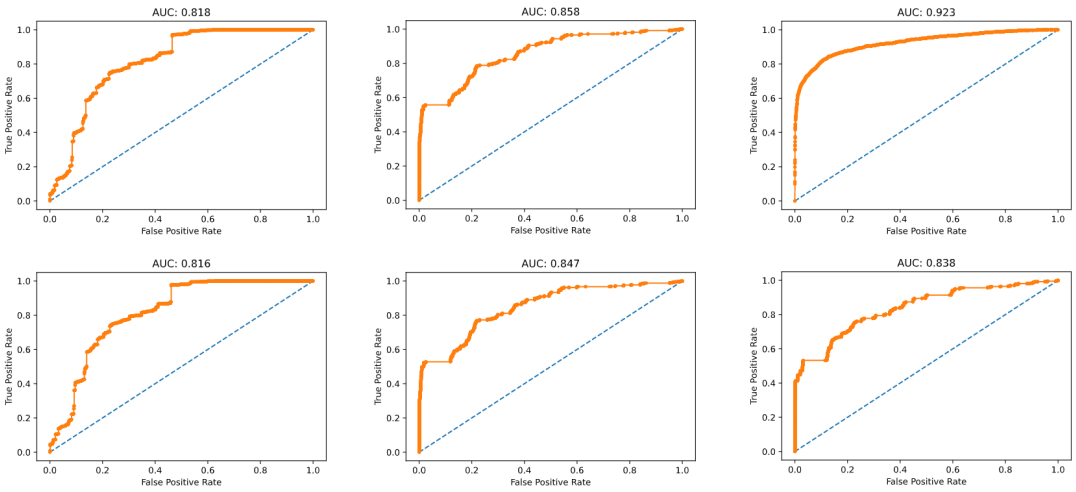

Figure 4.6: ROC curves derived from the training set (top row) and validation set (bottom row) for recording level models SNN_BCE_R (left column), SNN_CL_R (center column), and SNN_TL_R (right column).

are stacked in the input matrices. Translation-invariant patterns will be specific to individual audio features and therefore not recurring across feature borders.

The t-SNE representation of the embeddings from the training and validation sets can be seen in Figure 4.7. The 2D representation is, in this case, not a good performance measure since there are too few recordings in the validation set for the algorithm to form clusters. In the case of the training set, we can see that the models are able to separate the clusters with somewhat good results, however the distance between clusters is not as large as those from the frame level models.

The distributions of the pairwise distances from the validation set seen in Figure 4.8, also shown in Figures A.4 through A.6 in the appendix, show a similar pattern to those of the frame level models. However, it seems as though the recording level models are not as proficient at recognising pairs from differing classes. The distributions for these distances are shifted towards zero, more so than their frame level counterparts.

When making predictions on recording level, recording level data can also be utilised. The mean and standard deviations of the features can be computed and fed to the network in order to provide additional information. The recording level model SNN_BCE_R was extended to accept an additional data vector containing feature means and standard deviations. This vector entered the network as a separate branch concatenated to the flattened output from the convolution layer of the network. We name this network architecture SNN_BCE_R_MULTI. Similar hyperparameters were utilised during the training of the SNN_BCE_R_MULTI network. These can be seen in Table 4.6.

After training the models, they were evaluated on the validation set with identical user distribution as with previous models. The ROC curves for both the test set and validation set can be seen in Figure 4.9. Similar AUC scores for training and validation sets indicates that there is little overfitting in the model. This model architecture obtains the highest AUC score out of all the different models tested in this thesis.

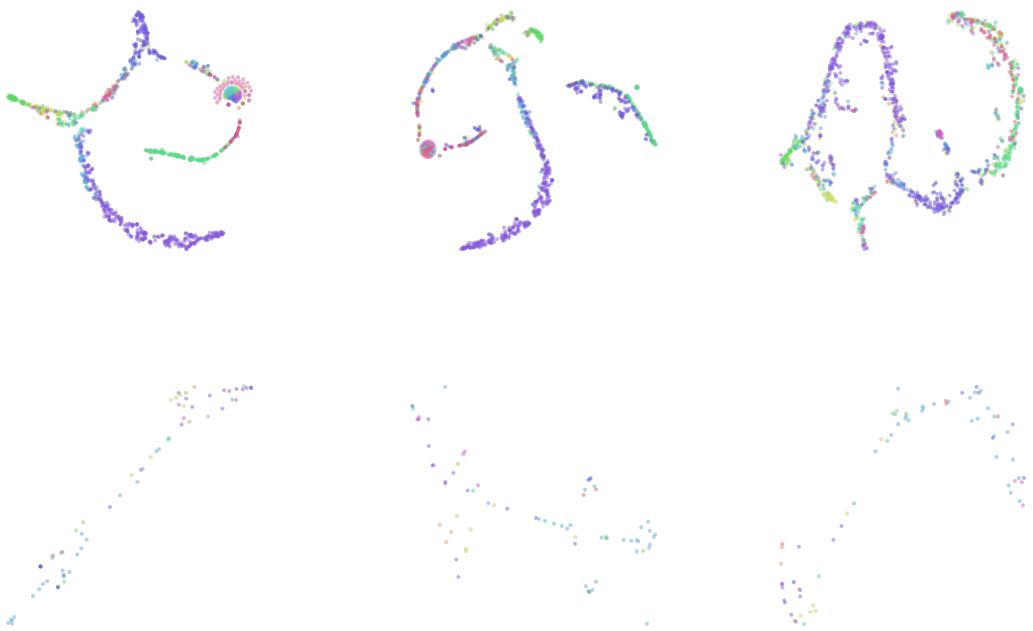

Figure 4.7: t-SNE visualisation of embeddings derived from the training set (top row) and validation set (bottom row) for recording level models SNN_BCE_R (left column), SNN_CL_R (center column), and SNN_TL_R (right column).

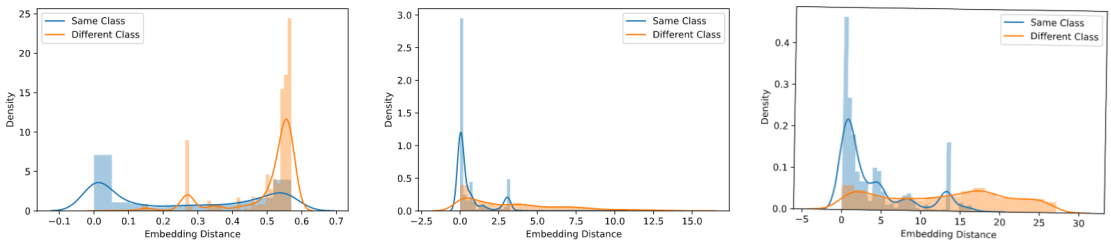

Figure 4.8: Distributions for pairwise same and differing class distances for recording level models SNN_BCE_R (left), SNN_CL_R (center), and SNN_TL_R (right).

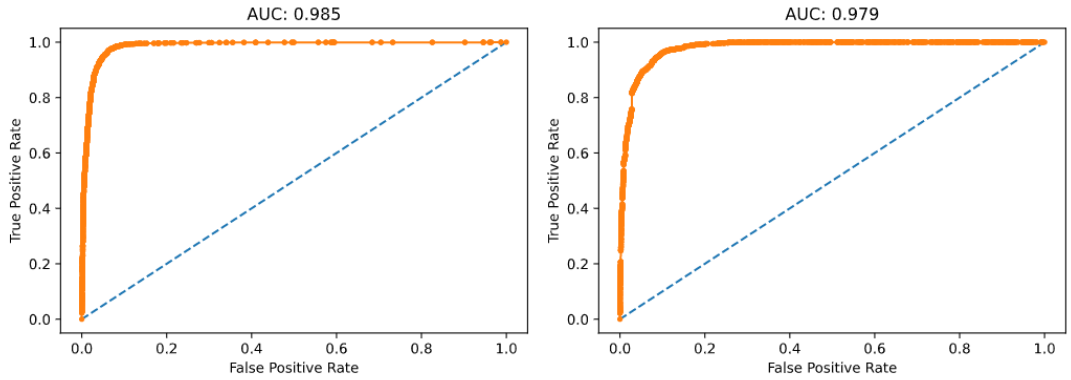

Figure 4.9: ROC curves and associated AUC scores for SNN_BCE_R_MULTI model on train set (left) and validation set (right).

HP: SNN_BCE_R_MULTI	
Learning rate	0.0001
Regularization	0.1
Batch size	12
Steps per epoch	500
Epochs	35

Table 4.6: Hyperparameters used during the training of multi input recording level model SNN_BCE_R_MULTI.

Although the AUC scores indicate that the model is highly capable of separating classes, the t-SNE visualisation of the embeddings shown in Figure 4.10 show less inter-class separation and clustering than most of the frame level models tested. The larger spread of points within clusters could be due to the fact that the data now comes from separate recordings as opposed to frames from within the same recordings.

Figure 4.10: t-SNE visualisation of embeddings derived from the training set (left) and validation set (right) for multi input recording level model SNN_BCE_R_MULTI.

Figure 4.11 shows the distributions of the pairwise distances from the validation set. There seems to be good separation between the distributions, and little overlap between density functions.

4.3 Model Comparison: Test Set

One frame level model (SNN_CL_F), and one recording level model (SNN_CL_R) was tested using recordings from the four individuals comprising the test set. The distance of the frames and recordings from the anchor embeddings for the different test subjects can be seen in Figures 4.12 and 4.13.

There is no guarantee that these patients did in fact experience a voice change during

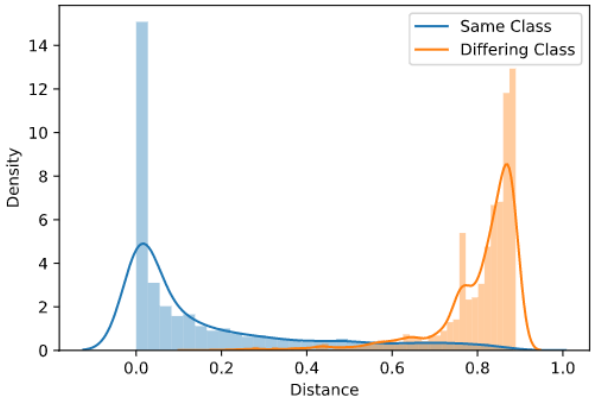

Figure 4.11: Distributions for pairwise same and differing class distances for the SNN_BCE_R_MULTI model.

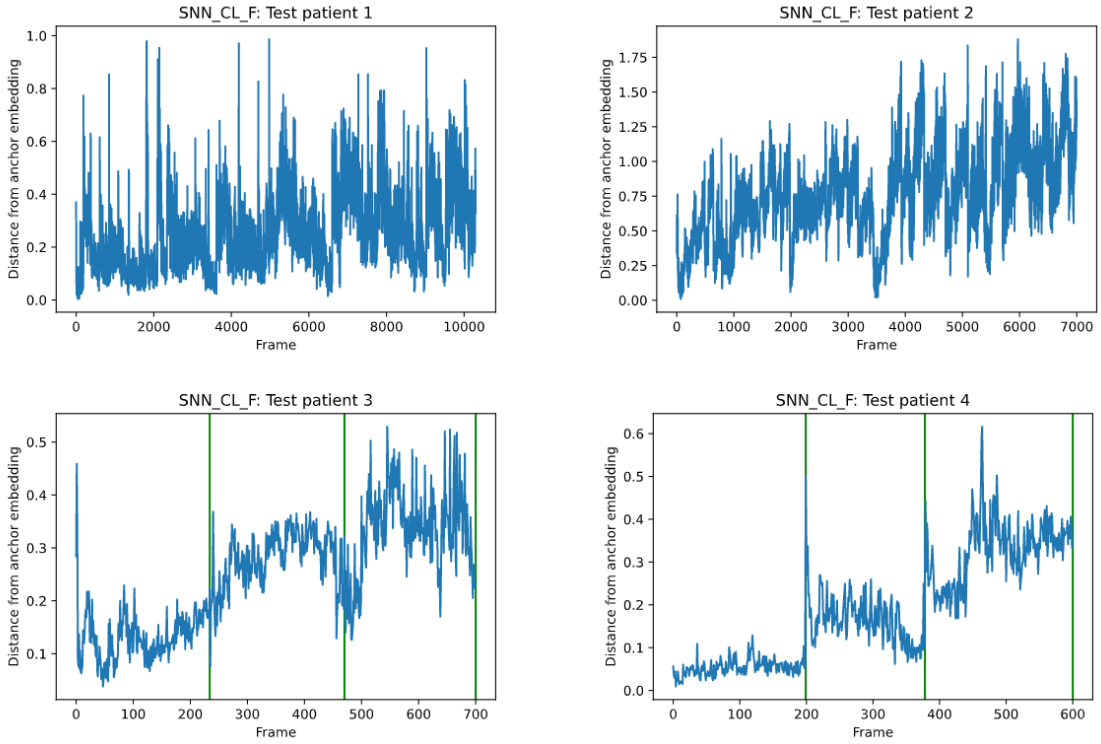

Figure 4.12: Euclidean distance between frame embeddings and anchor embedding over time for the four test patients. Green vertical line indicates separate recording instances.

the time period in question. The four subjects were chosen based on their higher likelihood of voice change due to their existing conditions. The plots in Figure 4.12 and 4.13 therefore need to be interpreted critically.

Looking at the plots for the frame level model (SNN_CL_F) in Figure 4.12, they indicate that the voices of all four subjects changed during the given time frame. The distances as measured from the anchor embedding gets progressively larger for all patients. Test patient 3 and 4 had fewer recordings and their respective diagrams cover a shorter time period than those of test patients 1 and 2 which could explain the smaller

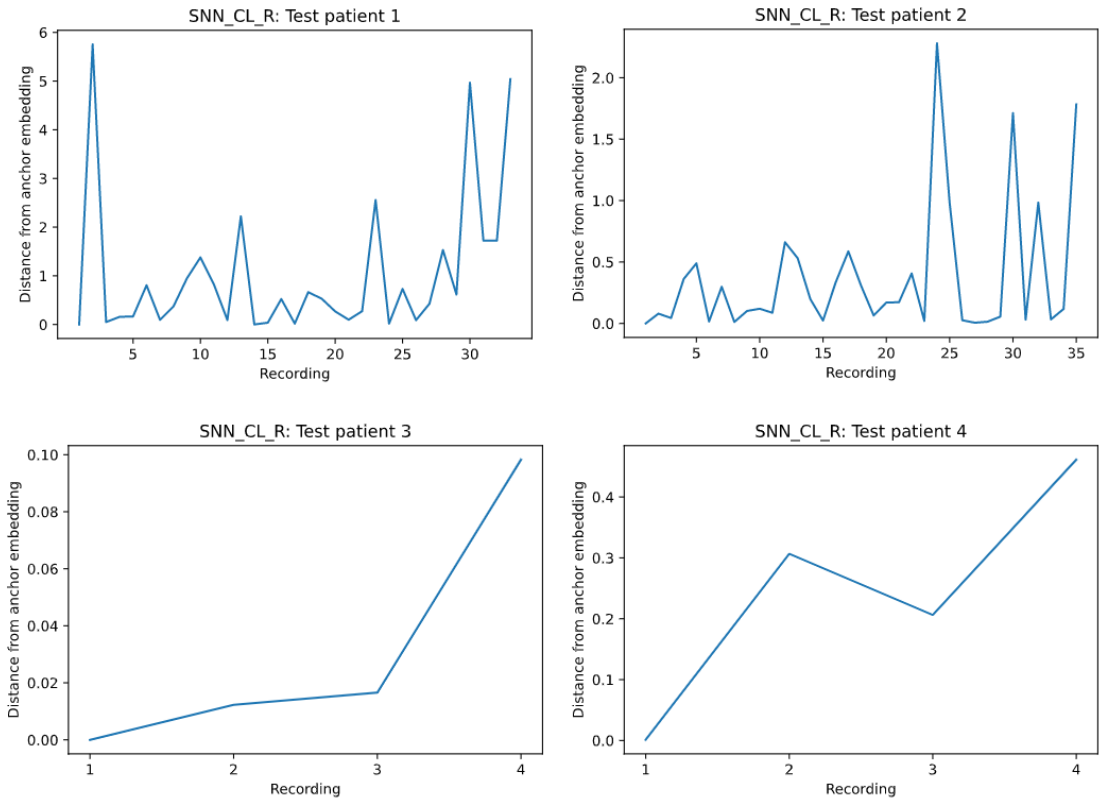

Figure 4.13: Euclidean distance between recording embeddings and anchor embedding over time for the four test patients.

increase in distance from their original anchor embedding. Test patient 4 displays clear signs of voice change and we can distinguish the separate recording instances as plateaus in the diagram. This is a phenomenon not observed as clearly for the other test patients.

The recording level model (SNN_CL_R) in Figure 4.13 seems to fluctuate more with regard to its classifications. The outputs from test patients 3 and 4 indicate that the distance from the anchor embedding to the recording is getting larger with time. However, for test patient 3, the magnitude of that distance is very small. It is difficult to distinguish any pattern from the outputs from test patients 1 and 2, perhaps a slight distance increase in patient 2. This forces us to the belief that the recording level models may not be reliable as a diagnostic means.

5 Conclusion

In this thesis, we have investigated the possibility of using Siamese Neural Network architectures in order to distinguish between voices, and detect voice changes over time. Due to the relatively little test data used to verify if the models could in fact detect any such changes, and the large degree of uncertainty whether the patients of the test set experienced a voice change, it is difficult to confirm with any degree of certainty if the hypothesis can be confirmed or not. However, high AUC scores, good clustering abilities, and separating distributions between alike and differing classifications, all point to the fact that the models are able to detect changes in a person's voice over time.

The second question investigated during this thesis was regarding the optimal model architecture and loss function to be used in the networks. The performance of the models were assessed according to primarily their AUC scores, but also their clustering abilities and output distributions. The frame level model which performed best with regard to these metrics was the bi-network structure utilising contrastive loss (SNN_CL_F), which achieved an AUC score of 0.950 on the validation set. The top performing recording level model was the bi-network structure with two input branches and a binary cross entropy loss function (SNN_BCE_R_MULTI) achieving an AUC score of 0.979 on the validation set.

The optimal audio features in terms of performance varied between the frame level and recording level models. The performance of the frame level models generally increased with more features provided to them, whilst the recording level models performed at their best when fed with single features. This only applied to the recording level models in cases when the input was provided as part of the same branch. The model with bi-branched inputs was the highest performing model overall in terms of the AUC score. The optimal features for all frame level models were the Mel spectrum, MFCCs, HNR, pitch, smoothed pitch, and AR-coefficients. The optimal features for the single-branch input recording level models was the MFCCs, and for the bi-branch input model the MFCCs combined with the pitch, HNR, AR-parameters, MFCCs, jitta, jitt, Shim, RAP, PPQ5, and APQ3.

6 Further Work

First of all, in the future, the models will need to be further tested on more data in order to verify their predictive powers. This will not be difficult as there will likely be more recordings available from patients with a recurrence of their cancer.

Furthermore, more training data may lead to better performances across all model architectures. Access to another dataset, or the continued collection of recordings would benefit all model types, but most of all the recording level models which had access to significantly fewer samples to train on. With more data, it may also be feasible to be more selective in the sampling of the data with regards to categories of features such as gender and benign illnesses affecting the voice. This would probably remedy any potential bias the current models have towards the current dataset.

There was not enough time available to train the models on the more phonetically varied and complex text-recordings (recording-T). It would be interesting to use features from these recordings during training as it will most likely lead to differing, and perhaps improved, results. When using the text-recordings, other features, such as percentage of pitched frames, can be used as inputs to the models. This approach also opens up the possibility of using public speech datasets such as the Librispeech dataset [24]. It is worth noting that the text-recordings in the current dataset are made in both Swedish and English, depending on the user who made the recording. The language in which the recordings were made can introduce further bias in the model.

Lastly, the networks can also be trained in a more selective manner where the batches are chosen according to classification difficulty. This would enable the network to learn and become more proficient at distinguishing between those users which were difficult to separate in the t-SNE plots.

Bibliography

[1] *Worldwide cancer data*. https://www.wcrf.org/dietandcancer/cancer-trends/worldwide-cancer-data. Accessed: 2021-03-04.

[2] *aryngeal and Hypopharyngeal Cancer: Statistic*. https://www.cancer.net/cancer-types/laryngeal-and-hypopharyngeal-cancer/statistics. Accessed: 2021-03-04.

[3] Facundo Bre, Juan Gimenez and Víctor Fachinotti. "Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks". In: *Energy and Buildings* 158 (Nov. 2017). DOI: 10.1016/j.enbuild.2017.11.045.

[4] Hojun Kim, Sungwook Park, In Gab Jeong, Sang Hoon Song, Youngdo Jeong, Choung-Soo Kim and Kwan Hyi Lee. "Noninvasive Precision Screening of Prostate Cancer by Urinary Multimarker Sensor and Artificial Intelligence Analysis". In: *ACS Nano* 0.0 (0). PMID: 33296173, null. DOI: 10.1021/acsnano.0c06946. eprint: https://doi.org/10.1021/acsnano.0c06946. URL: https://doi.org/10.1021/acsnano.0c06946.

[5] Gregory R. Koch. "Siamese Neural Networks for One-Shot Image Recognition". In: 2015.

[6] P. Manocha, R. Badlani, A. Kumar, A. Shah, B. Elizalde and B. Raj. "Content-Based Representations of Audio Using Siamese Neural Networks". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 3136–3140. DOI: 10.1109/ICASSP.2018.8461524.

[7] Diego Droghini, Fabio Vesperini, Emanuele Principi, Stefano Squartini and Francesco Piazza. "Few-Shot Siamese Neural Networks Employing Audio Features for Human-Fall Detection". In: *Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence*. PRAI 2018. Union, NJ, USA: Association for Computing Machinery, 2018, 63–69. ISBN: 9781450364829. DOI: 10.1145/3243250.3243268. URL: https://doi.org/10.1145/3243250.3243268.

[8] Hao Xiong, Peiliang Lin, Jin-Gang Yu, Jin Ye, Lichao Xiao, Yuan Tao, Zebin Jiang, Wei Lin, Mingyue Liu, Jingjing Xu, Wenjie Hu, Yuewen Lu, Huaifeng Liu, Yuanqing Li, Yiqing Zheng and Haidi Yang. "Computer-aided diagnosis of laryngeal cancer via deep learning based on laryngoscopic images". In: *EBioMedicine* 48 (2019), pp. 92–99. ISSN: 2352-3964. DOI: https://doi.org/10.1016/j.ebiom.2019.08.075. URL: https://www.sciencedirect.com/science/article/pii/S2352396419305973.

[9] HyunBum Kim, Juhyeong Jeon, Yeon Jae Han, YoungHoon Joo, Jonghwan Lee, Seungchul Lee and Sun Im. "Convolutional Neural Network Classifies Pathological Voice Change in Laryngeal Cancer with High Accuracy". In: *Journal of Clinical Medicine* 9.11 (2020). ISSN: 2077-0383. DOI: 10.3390/jcm9113415. URL: https://www.mdpi.com/2077-0383/9/11/3415.

[10] Jorge Prieto Rubio. *DISEÑO DE UN CONVERSOR ANALÓGICO DIGITAL CON TOPOLOGÍA WILKINSON EN TECNOLOGÍA CMOS 0.35 μm*. 2019.

[11] *Music Feature Extraction in Python*. https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d. Accessed: 2021-03-05.

[12] Georg Lindgren, Holger Rootzén and Maria Sandsten. *Stationary Stochastic Processes for Scientists and Engineers*. Boca Raton: Taylor and Francis Group, 2014.

[13] James Cooley and John Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Mathematics of Computation* 19.90 (1965), pp. 297–301.

[14] *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between*. https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html. Accessed: 2021-03-24.

[15] A. de Cheveigné and Hideki Kawahara. "YIN, a fundamental frequency estimator for speech and music." In: *The Journal of the Acoustical Society of America* 111 4 (2002), pp. 1917–30.

[16] João Paulo Teixeira and André Gonçalves. "Algorithm for Jitter and Shimmer Measurement in Pathologic Voices". In: *Procedia Computer Science* 100 (2016). International Conference on ENTERprise Information Systems/International Conference on Project MANagement/International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2016, pp. 271–279. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.09.155. URL: https://www.sciencedirect.com/science/article/pii/S1877050916323237.

[17] João Paulo Teixeira, Carla Oliveira and Carla Lopes. "Vocal Acoustic Analysis – Jitter, Shimmer and HNR Parameters". In: *Procedia Technology* 9 (2013). CENTERIS 2013 - Conference on ENTERprise Information Systems / ProjMAN 2013 - International Conference on Project MANagement/ HCIST 2013 - International Conference on Health and Social Care Information Systems and Technologies, pp. 1112–1122. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2013.12.124. URL: https://www.sciencedirect.com/science/article/pii/S2212017313002788.

[18] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[19] *Understanding Activation Functions in Deep Learning*. https://learnopencv.com/understanding-activation-functions-in-deep-learning/. Accessed: 2021-03-02.

[20] *2D Convolution block*. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-convolution-block/. Accessed: 2021-03-02.

[21] *2D Convolution block*. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/1d-convolution-block. Accessed: 2021-03-04.

[22] *Variational Autoencoders are Beautiful*. https://www.compthree.com/blog/autoencoder/. Accessed: 2021-03-03.

[23] *VoiceDiagnostic*. https://www.voicediagnostic.com/. Accessed: 2021-03-09.

[24] V. Panayotov, G. Chen, D. Povey and S. Khudanpur. "Librispeech: An ASR corpus based on public domain audio books". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

Appendix A

A

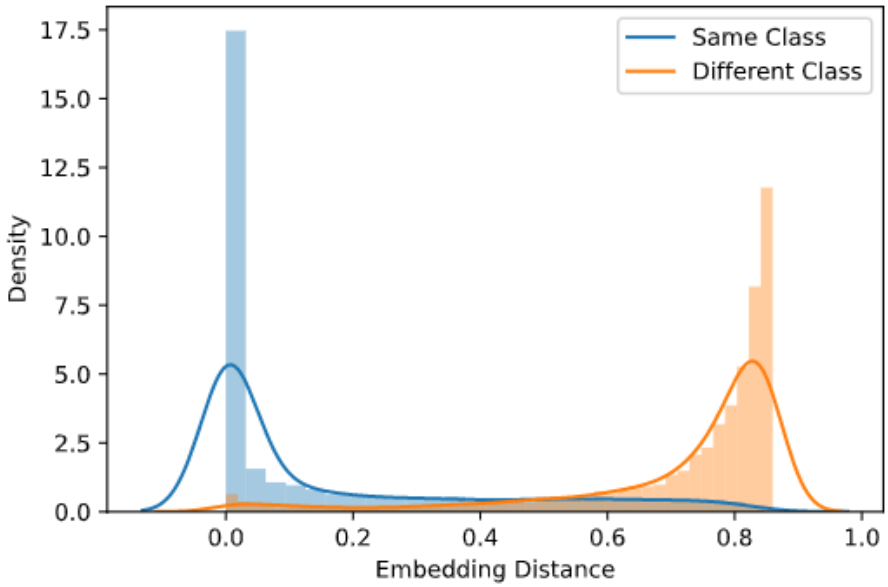

Figure A.1: Distributions for pairwise same and differing class distances for the frame level model SNN_BCE_F.

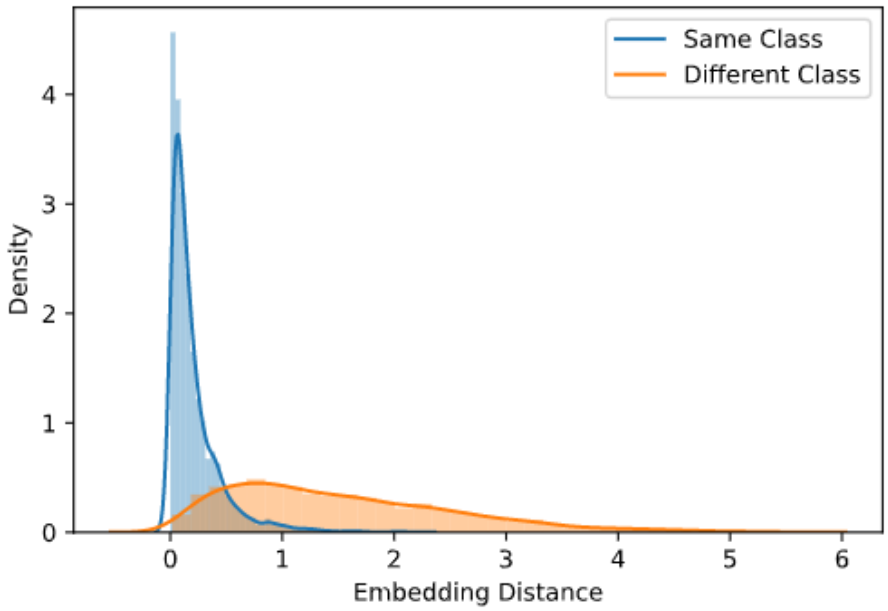

Figure A.2: Distributions for pairwise same and differing class distances for the frame level model SNN_CL_F.

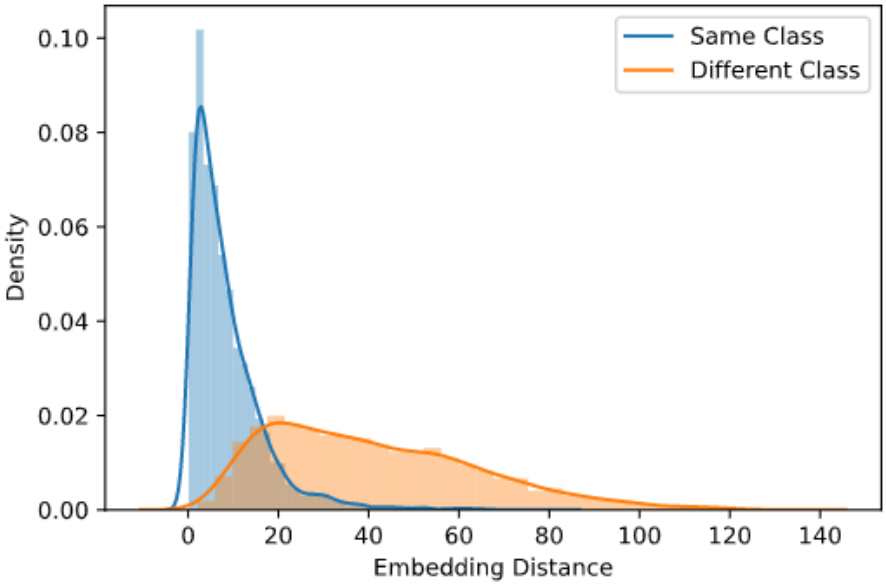

Figure A.3: Distributions for pairwise same and differing class distances for the frame level model SNN_F.

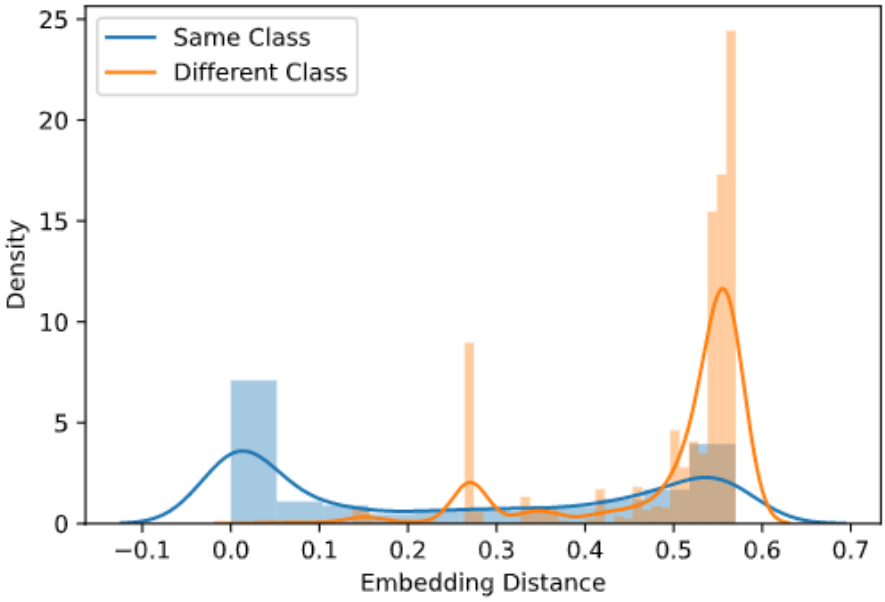

Figure A.4: Distributions for pairwise same and differing class distances for the recording level model SNN_BCE_R.

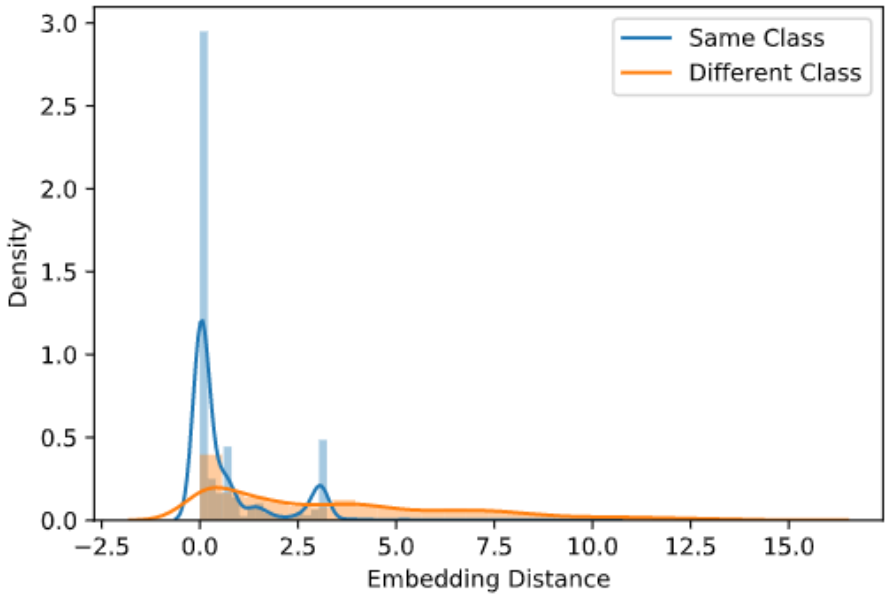

Figure A.5: Distributions for pairwise same and differing class distances for the frame level model SNN_CL_R.

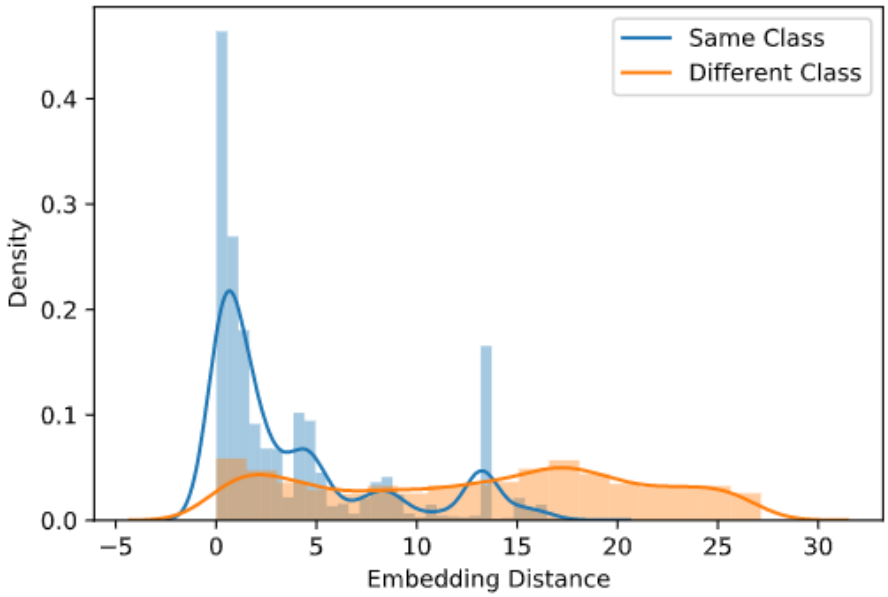

Figure A.6: Distributions for pairwise same and differing class distances for the frame level model SNN_R.

Master's Theses in Mathematical Sciences 2021:E18
ISSN 1404-6342

LUTFMS-3414-2021

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

http://www.maths.lth.se/