UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

Exploring the use of Contrastive Learning to perform Audio Classification for TB Screening.

Minette Farrell
23566892

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: Prof T. Niesler
Department of Electrical and Electronic Engineering

November 6, 2023

Acknowledgements

I want thank to Professor Thomas Niesler, for all the advice and feedback during the semester. I would also like to thank the DSP lab for all the support and coffee. Specifically, I want to thank Michael for the code he provided to help me get going, Julian and Christiaan for all the time spent helping me get to terms with the difficult concepts for this project, and Rachel and Paul for all the Chalkboard trips and company during the difficult times. I would also like to thank my family for all the support (and food) provided during this time.

UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

Studentenommer / *Student number* **23566892**	Handtekening / *Signature* MFarrell
Voorletters en van / *Initials and surname* **M. Farrell**	Datum / *Date* **November 6, 2023**

Abstract

English

Objective: Tuberculosis (TB) is the leading cause of death caused by an infectious disease. In developing countries, many patients with tuberculosis are undiagnosed due to difficult circumstances. An ongoing research project at the University of Stellenbosch is working towards implementing a machine-learning model that can be used for automatic TB screening. This project aims to extend this research to determine whether contrastive learning can be used to improve the performance and robustness of a model for TB screening.
Approach: This project considers different ways to implement contrastive learning by considering various image augmentation processes, contrastive loss functions, and architectures, such as SimCLR, COLA, and SimSiam.
Main results: The test results from this project indicate that contrastive learning performs well under certain circumstances for this dataset, but it does not generalise well. Overall, the models trained using NT-Xent loss performed the best. The results also show that a Siamese network can be used to generate features for logistic regression.
Significance: This project shows that there is potential in applying contrastive learning to perform automatic TB classification.

Afrikaans

Doel: Tuberkulose is die hoof oorsaak van dood wat veroorsaak word deur 'n aansteeklike siekte. Baie pasiënte in ontwikkelende lande word nie betyds gediagoniseer met tuberkulose nie as gevolg van verskillende bydraende faktore. Die doel van hierdie projek is om te identifiseer of kontrasterende afrigmetodes gebruik kan word om 'n model te leer om pasiënte te klassifiseer gebasseer op 'n hoes opname.
Metode: Die projek kyk na verskillende metodes wat gebruik kan word om kontrasterende afrigmetodes te implementeer insluitend verskeie beeld transformasies, verlies funksies en model benaderings. *Resultate:* Die toets resultate dui daarop aan dat kontrasterende afrigmetodes onder sommige omstandighede goed presteer, maar dat dit nie goed veralgemeen nie. Ingeheel dui die resultate daarop aan dat genormaliseerde temperatuur-geskaalde kruisentropieverlies die beste vaar tussen die twee verlies funksies en dat 'n Siamese netwerk gebruik kan word om kenmerke te genereer vir 'n logistiese regressiemodel.
Gevolgtrekking: Hierdie projek wys dat die toepassing van kontrasterende leer vir tuberkulose diagnose belowend is.

Contents

List of Figures

List of Tables

Nomenclature

Variables and functions

D_w	Euclidean distance between two points.
$\ell_{i,j}$	Contrastive loss function.
$sim(z_i, z_j)$	Similarity between z_i and z_j.

Acronyms and abbreviations

TB	Tuberculosis
PTB	Pulmonary Tuberculosis
MFCC	Mel-frequency Cepstral Coefficients
ZCR	Zero Crossing Rate
LR	Logistic Regression
kNN	K-Nearest Neighbour
SVM	Support Vector Machine
MLP	Multilayer Perceptron
CFN	Context-free Network
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
Resnet	Residual-based Neural Network Architecture
SimCLR	Simple Framework for Contrastive Learning of Visual Representations
BYOL	Bootstrap Your Own Latent
SwAV	Swapping Assignments between multiple Views of the same image
SimSiam	Simple Siamese Network
COLA	COntrastive Learning for Audio
CLAR	Contrastive Learning of Auditory Representation
NT-Xent	Normalized Temperature-scaled Cross Entropy Loss
NCE	Noise Contrastive Estimation
TST	Tuberculin Skin Test
IGRA	Interferon-gamma Release Assay
CT	Computed Tomography
COVID	Corona Virus Disease
DSP	Digital Signal Processing
DFT	Discrete Fourier Transform
BCE	Binary Cross Entropy
AUC	Area under the curve

Chapter 1

Introduction

According to the World Health Organization, tuberculosis (TB) is a bacterial disease that affects 10 million people and causes 1.5 million deaths per year. It is caused by bacteria (Mycobacterium tuberculosis) and it is spread when people with pulmonary TB cough, sneeze, or spit. Even though TB is curable, millions of TB patients (especially in developing countries) do not receive treatment due to a lack of medicinal facilities, and healthcare staff, a shortage of treatments, and undiagnosed patients [1].

1.1. Purpose of this study

There are currently two kinds of screening tests available to diagnose TB, namely the Mantoux tuberculin skin test (TST) and a blood test known as interferon-gamma release assay (IGRA) [2]. According to Public Health Action, before the COVID-19 pandemic, 2.9 million of the 10 million people who contracted TB were not diagnosed or reported. This indicates that there are many cases of TB infections that are missed, leading to more bacterial transmissions and deaths. Some of the reasons for missed cases include prioritising other health conditions, high patient loads, communication challenges, patients refusing to receive treatment, maltreatment of the staff, and the stigma relating to having TB [3].

An ongoing research project at the University of Stellenbosch aims to provide developing countries with an accessible alternative to diagnose TB. The aim is to analyse cough recordings to determine whether that person has TB or not. This would be a simple and effective test that can be carried out at primary healthcare clinics without the need for specialised staff or laboratory facilities. This can help address some of the challenges that developing countries face to decrease the number of undetected TB cases. The research group has found that machine learning models such as logistic regression (LR) or a convolutional neural network (CNN) can classify TB using cough audio recordings [4], [5], [6] and [7].

1.2. Problem statement

This project aims to expand on the current research, by investigating whether contrastive learning can be used to improve an existing model's accuracy and robustness. This will be done by implementing a Siamese network with a contrastive loss function to generate features that can be used as input data for the existing model.

1.3. Significance and motivation

This project aims to improve the existing model's performance and robustness to external noise. Improving the model's empirical performance increases the likelihood of the model being accepted by the medical profession to assist in diagnosing TB patients. Improving the model's robustness can lead to a model that performs better under different circumstances.

1.4. Limitations

This project has access to a small dataset consisting of a few patients' cough recordings since each recording has to be done by a trained healthcare worker in a clinic. Each cough also needs to be pre-processed and annotated. The laboratory analysis of sputum that is used to determine whether a patient has TB or not is expensive and time-consuming. To address the limitations of a small dataset, the existing model uses nested cross-validation. This project will also use Siamese networks and data augmentation processes to generate more data for training.

1.5. Brief chapter overview

The structure of the remainder of this paper is as follows. Chapter 2 will discuss some of the theory concepts used throughout the paper. Chapter 3 will look at existing literature relating to this project. Chapter 4 will explain the data collection and processing steps taken. Chapter 5 will explain the approach that was followed and the model implementation. Chatper 6 will discuss the results of the models. Chapter 7 summarises the results and findings.

Chapter 2

Background information

This chapter will discuss and explain some of the concepts that is used in this project. This includes a list of definitions, a summary of self-supervised learning, a discussion on contrastive learning, and an explanation of a basic Siamese network and contrastive learning architectures. Lastly, different loss functions that are used in contrastive learning will be discussed.

2.1. Definitions

Representation

The word representation is used in the literature to refer to the output of an encoder such as a ResNet model. In the context of this project, representation refers to the embedding of input data to a new space (which could be lower or higher dimensional), that encapsulates the characteristics of the data in a format that is more useful for analysis or further processing than the input data itself.

Augmentation

The input for this project consists of audio samples that are converted into mel spectrogram images. In order to contrast the true input, each image is augmented to create an augmented image. The augmentation processes take the mel spectrogram image and apply one or more image augmentation techniques to each image. Image augmentation techniques that are investigated include fade in/out, time masking, frequency masking, time shifting, time stretching, uniform frequency offset and amplitude compression.

2.2. Self-supervised learning

Self-supervised learning is a machine learning technique where the model uses unlabeled data for training to differentiate between parts of the input data. This is also referred to as pretext learning or predictive learning. The idea behind self-supervised learning is to address the challenge posed by the usually limited amount of labeled data available. The model solves unsupervised problems by implicitly generating representations that can be used as labels that describe the input data in a way that encodes different characteristics of the input data.

2.3. Contrastive learning

Contrastive learning is a self-supervised learning method first suggested by Mikolov et al. [8] for natural language processing in 2013. Contrastive learning aims to learn low-dimensional representations of high-dimensional input data with the aim of having similar inputs close to each other in the representation space. This can be done by using an architecture known as a Siamese network and a contrastive loss function to minimises the distance between similar images.

2.4. Siamese network

A Siamese network is a neural network that consists of two sub-networks, where one sub-network is trained and the other sub-network shares the trained sub-network's parameters and weights. Figure 2.1 shows the layout of a basic Siamese network where the two sub-networks are shown as parallel branches. Typically, the input for this model will be a high dimensional representation, such as a mel spectrogram image, while the features generated will be a low dimensional representation. A basic Siamese network consists of two identical sub-networks that share the same weights. Each sub-network consists of an encoder $f(\cdot)$, which returns a representation of the input, followed by a projection head $g(\cdot)$ which returns the features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$ [9].

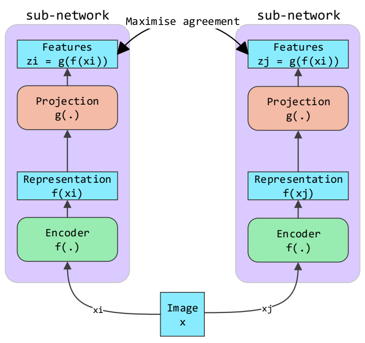

Figure 2.1: A general Siamese network takes in an image $\boldsymbol{x}$, creates a copy of the original input image $\boldsymbol{x_i}$ and an augmentation of the original image $\boldsymbol{x_j}$. Each sub-network consists of an encoder $f(\cdot)$, and a projection head $g(\cdot)$ which generates features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. The loss function uses the features generated to maximise the agreement between the two sub-networks.

Four popular Siamese networks used to perform contrastive learning are shown in Figure 2.2, namely SimCLR, BYOL, SwAV, and SimSiam. A short description of each architecture is given below (a detailed discussion of each architecture is given in Section 3.2).

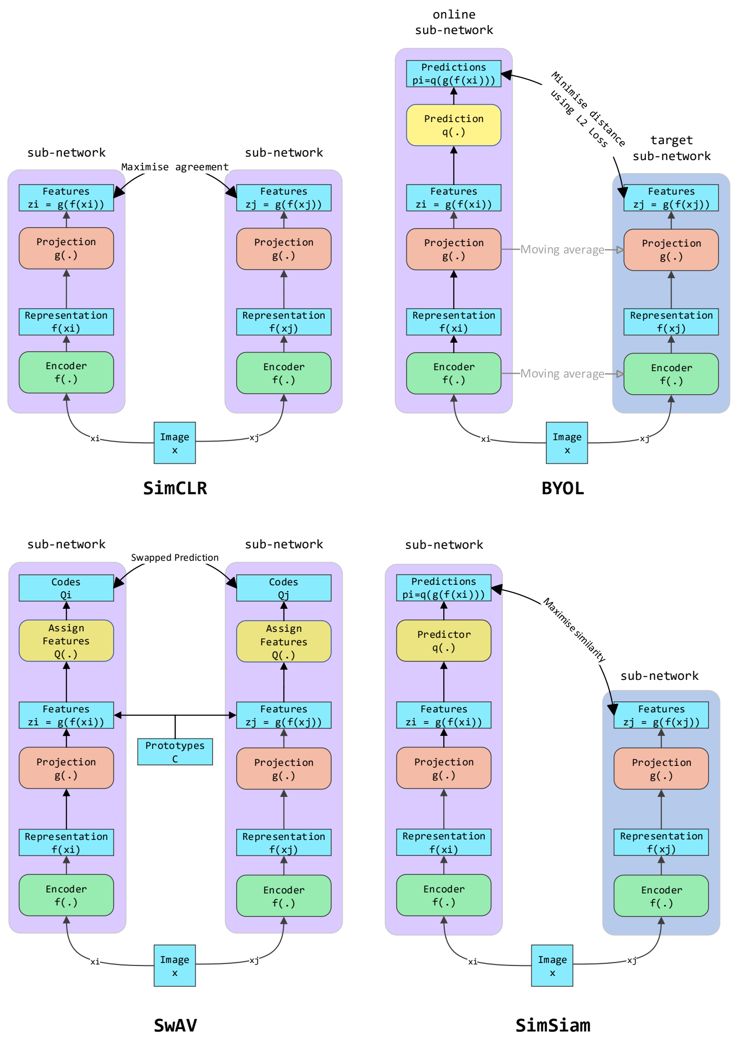

Figure 2.2: Four different Siamese architectures, SimCLR, BYOL, SwAV, and SimSiam are shown to highlight the differences between the four architectures. SimCLR uses positive and negative pairs to create contrast, BYOL uses a momentum encoder to update the weights of the target network, SwAV uses online clustering and the Sinkhorn Knopp algorithm, and SimSiam uses gradient stopping to prevent collapsing.

Gradient collapse: Gradient collapse occurs when the gradients of the loss function used in Siamese networks become very small. This is a common phenomenon in contrastive learning where loss functions are used to minimise the distance between similar images. Each of the architectures discussed below has a unique way of ensuring that the architecture does not collapse.

Siamese Networks for image classification

SimCLR: Compared to the basic Siamese network shown in Figure 2.1, SimCLR uses both a positive and a negative pair of input images to prevent gradient collapse (basic Siamese networks use only one positive pair). SimCLR uses a contrastive loss function to minimise the distance between similar images and maximise the distance between dissimilar images in the representation space.

BYOL: BYOL uses two sub-networks known as the target and online sub-networks. Instead of having two identical encoders, the online sub-network has a basic encoder, and the target sub-network has a momentum encoder. The momentum encoder updates the weights of the target sub-network based on a moving average of the weights of the online sub-network to prevent gradient collapse (both encoders usually share the same weights for a basic Siamese network).

SwAV: SwAV uses online clustering and a new image processing technique known as multi-crop to prevent gradient collapse. Multi-crop takes one image and creates multiple views of the image that are lower resolution. SwAV also uses the Sinkhorn Knopp algorithm, to minimise the distance between similar images. Sinkhorn Knopp is an algorithm that uses entropy regularization and iterative scaling to find a near-optimal solution that aligns two probability distributions, whilst minimising the transportation cost [10].

SimSiam: SimSiam shows that the aspects implemented in the other architectures to prevent collapsing can be removed without causing the model to collapse by implementing gradient stopping. The Siamese network is modified to include a predictor on one of the sub-networks, making it asymmetric. The weights of the sub-network with the prediction layer are updated, while the weights of the sub-network without the prediction layer remain unchanged to prevent the model from collapsing. The authors of SimSiam refer to it as: "SimCLR without the negative pairs", "SwAV without online clustering" and "BYOL without the momentum encoder" [11].

Siamese Networks for audio classification

This section will briefly introduce two audio-specific contrastive learning architectures. A more comprehensive summary of each architecture is given in Section 3.2.

COLA: COLA is a general-purpose audio representation model. It uses audio signals as input data. The audio is split into segments to form the positive pairs. This architecture uses bi-linear similarity instead of cosine similarity to measure the distance between positive pairs.

CLAR: CLAR also takes audio signals as input data. It compares two sub-networks, one that uses an augmented audio signal as input and another that uses a spectrogram of the augmented audio as input. The loss function is a combination of cross-entropy loss and contrastive loss. CLAR uses different data augmentation techniques specific to audio classification such as time masking and fade in/out create the best representations.

2.5. Loss functions

This section will briefly describe the different loss functions found in the contrastive learning literature. These functions include normalised temperature-scaled cross-entropy loss, contrastive loss, and triplet loss. Other loss functions that have been proposed include lifted structured loss, n-pair loss, noise contrastive estimation (NCE), InfoNCE, and soft-nearest neighbors loss [12].

Normalised temperature-scaled cross-entropy loss

Normalised temperature-scaled cross-entropy loss (NT-Xent) is based on the softmax function with the addition of a temperature normalisation factor τ and vector similarity function $sim(\boldsymbol{z_i}, \boldsymbol{z_j})$:

$$\ell_{i,j} = -log\left(\frac{\exp\left(sim(\boldsymbol{z_i}, \boldsymbol{z_j})/\tau\right)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp\left((sim(\boldsymbol{z_i}, \boldsymbol{z_k}))/\tau\right)}\right) \tag{2.1}$$

In Equation (2.1), $\mathbb{1}_{[k\neq i]} \subseteq 0, 1$ is an indicator function equal to 0 if $k = i$ (the same image), otherwise it evaluates to 1, τ is the temperature factor (default 0.5) and $\boldsymbol{z_i} = g(f(\boldsymbol{x_i}))$ refers to the features generated, where $g(\cdot)$ is the output of the projection head and $f(\cdot)$ is the output of the encoder (usually ResNet50) as seen in Figure 2.1. The similarity function $sim(\boldsymbol{z_i}, \boldsymbol{z_j})$ uses the cosine distance between vectors in the representation space:

$$sim(\boldsymbol{z_i}, \boldsymbol{z_j}) = \frac{\boldsymbol{z_i}^T \boldsymbol{z_j}}{||\boldsymbol{z_i}|| \cdot ||\boldsymbol{z_j}||}$$

Contrastive loss

Contrastive loss is defined for a pair of samples $\boldsymbol{z_i}, \boldsymbol{z_j}$ and minimises the distance between similar samples and maximises the distance between dissimilar samples.

$$\mathcal{L} = (1 - Y) * ||\boldsymbol{z_i} - \boldsymbol{z_j}||^2 + Y * \max(0, m - ||\boldsymbol{z_i} - \boldsymbol{z_j}||)^2 \tag{2.2}$$

In Equation (2.2), m is a hyper-parameter known as the margin that determines how far the representations of different images have to be considered dissimilar, $\boldsymbol{z_i}$ is the output of the projection head $g(f(\boldsymbol{x_i}))$ where f is the encoder, g is the projection head and $\boldsymbol{x_i}$ is the i-th input image. Y is an indicator variable for which Y=0 when the images are considered to be similar and Y=1 when the images are considered dissimilar. When Y=0, the second term falls away, and the distance between the images, $||\boldsymbol{z_i} - \boldsymbol{z_j}||^2$ is minimised. When Y=1, the first term falls away, and $m - ||\boldsymbol{z_i} - \boldsymbol{z_j}||^2$ is minimised, i.e., the distance between the dissimilar images is maximised until the distance is equal to the margin m.

Triplet loss

Triplet loss is an extension of contrastive loss. Instead of pairs of samples, triplet loss is defined for a triplet of samples. This triplet consists of an anchor image (I), a positive image (I^+), and a negative image (I^-). Triplet loss encourages the distance between the anchor and the positive sample to be less than the distance between the anchor and the negative sample:

$$\mathcal{L} = max(0, ||\boldsymbol{z} - \boldsymbol{z}^+||^2 - ||\boldsymbol{z} - \boldsymbol{z}^-||^2 + m) \tag{2.3}$$

In Equation (2.3), m is the same hyper-parameter as above and $\boldsymbol{z_i}$ is the output of the projection head as described above. The triplet loss $\mathcal{L}$ reaches its minimum when the difference between the distance between the anchor and the positive sample and the distance between the anchor and the negative sample is less than the margin m.

2.6. Conclusion

This chapter discussed various theoretical concepts needed to understand contrastive learning and the terminology used throughout this project. The next chapter is a literature review of the existing work done in the field of TB classification and contrastive learning.

Chapter 3

Literature Review

This chapter aims to provide a brief overview of the existing literature with respect to the TB cough classification as well as an in-depth explanation of the existing architectures found in the literature.

3.1. Cough audio classification

The DSP Group at Stellenbosch University has been engaged in the development of cough sound classifiers for a number of years. In the following sections, the published results are reviewed.

The first paper aimed to identify whether one could use a machine learning model to accurately predict whether a patient has TB or not [4]. To classify patients, a logistic regression model was used. It found that classification of short-term spectral features is possible using logistic regression with an accuracy of 78%. The work also found that the classifier is using some spectral distinction beyond the limit of human auditory perception. This shows that this field of research has promise.

The research mentioned above established that it is possible to detect TB from a data set with people who are diagnosed with TB and healthy patients. To extend this, new data was collected to investigate whether one can detect TB among sick patients, but not necessarily with TB [6]. Mel-frequency cepstral coefficients (MFCCs), log-filterbank energies, zero-crossing rate, and kurtosis were considered as features. Logistic regression (LR), k-nearest neighbors (kNN), support vector machines (SVM), multi-layer perceptions (MLP), and a convolutional neural network (CNN) were considered as classifiers. The study found that logistic regression performed the best among the five classifiers considered. Nested cross-validation was used since the data set is still small. This paper shows that it is possible to distinguish TB from other lung illnesses.

During the COVID-19 pandemic, the research team decided to consider whether the existing research can be used to determine whether a patient has COVID-19 or not. The study described in [5] investigated whether voice recordings on a smartphone can be used to diagnose COVID-19 patients remotely. The same features used in [6] were examined. Seven machine learning algorithms were implemented: logistic regression (LR), k-nearest neighbor (KNN), support vector machine (SVM), multi-layer perceptron (MLP), convolutional neural network (CNN), long short-term memory (LSTM) and a residual-based neural network architecture (ResNet). The study found that the Resnet50 had the best performance, followed by the CNN and LSTM classifiers. Similar to [6], this paper found that the classifiers use some information imperceptible to the human ear to classify the input data. This paper shows that the Resnet50 classifier can accurately predict whether patients have COVID-19 using only a smartphone recording.

In 2022, researchers in India published an article that considers the development of an AI platform for pulmonary tuberculosis (PTB) screening using cough samples [13]. The data for this research, which was collected at Andhra Medical College in India, consists of 278 positive and 289 negative PTB patients. The data was split into training, validation, and testing sets using k-fold cross-validation. The algorithm used is based on a multi-modal convolution neural network and a tabular model based on the collected features. Mel-frequency cepstral coefficient (MFCC) spectrograms were used as input to the convolution neural network (CNN), which consists of a ResNet34 model. Primary and secondary features were used as input for the tabular model. The final layer of the model combines the logic of the CNN and the tabular model to perform classification using both models. The model achieved 86.82% accuracy, 90.36% sensitivity, and 84.67% specificity, meeting the minimum requirements for a medical diagnosis as set out by the World Health Organisation.

Another research group at the University of Mumbai in India considered tuberculosis detection using X-rays [14]. The group used a convolution neural network (CNN) to perform image classification. The model consists of four phases namely pre-processing, image segmentation, feature extraction, and classification. The dataset consists of X-ray images collected from a medical clinic in Shenzhen, China. The dataset consisted of 80 X-rays from patients without TB and 58 X-rays from patients who showed indications of TB. The CNN model consists of 3 hidden layers, 9 convolutional layers, 10 ReLU layers, 4 drop-out layers, and 3 max pooling layers. The group used an Adam optimiser to achieve an accuracy of 82.09%, a precision of 95%, and a recall of 85%.

3.2. Relevant work in contrastive learning

The idea of using contrastive measures to improve the performance of image processing models was first introduced by Mehdi Noroozi and Paolo Favaro [15]. Their work explored the idea of using a CNN to solve jigsaw puzzles using a context-free network (CFN). The main objective of this paper was to demonstrate that a CFN can be used as a pretext task to identify labels that contain semantically relevant content. This paper showed that using a Siamese context-free network can solve puzzles and has the potential to solve image processing problems using similar concepts.

SimCLR is a Siamese network that is specifically designed for image processing and is used to perform contrastive learning of visual representations [16]. The paper considered different aspects of what makes a good contrastive learning model. The framework consists of four major parts. Firstly, data augmentation - each input image is randomly transformed to form positive pairs $\boldsymbol{x_i}, \boldsymbol{x_j}$, and negative pairs $\boldsymbol{x_i}, \boldsymbol{x_{k \neq i,j}}$. The transformations consist of random cropping, random color distortions, and random Gaussian blur. Secondly, a base encoder $f(\cdot)$ extracts representation vectors from the augmented pairs, for example using a ResNet. Thirdly, a projection head $g(\cdot)$ consisting of a linear dense layer, followed by a ReLU activation function and another linear dense layer, maps the representation vectors into a space where the loss function can be applied. Lastly, a contrastive loss function $\ell_{i,j}$ is used to minimise the distance between similar images and maximise the distances between dissimilar images. SimCLR was trained on the ImageNet Large Scale Visual Recognition Challenge 2012 dataset. This dataset has over 1 million training images, 50,000 validation images, and 100,000 test images [17]. In their paper, the authors showed that data augmentation is crucial, that a nonlinear transformation (projection head) after the encoder, improves representation quality, that the contrastive loss function benefits from normalisation, and that contrastive learning benefits from larger batch sizes. Figure 3.1 shows the SimCLR architecture.

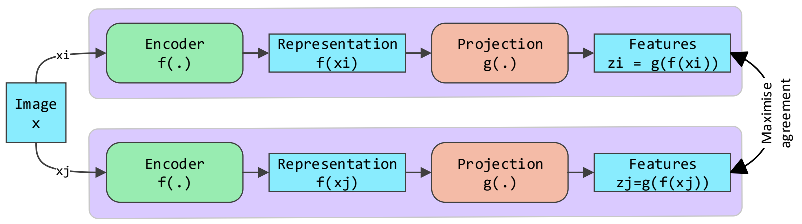

Figure 3.1: The SimCLR architecture takes an input image $\boldsymbol{x}$, creates two instances the anchor $\boldsymbol{x_i}$ and the augmentation $\boldsymbol{x_j}$. An encoder $f(\cdot)$ is used to create representation $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$. These representations then pass through a projection head $g(\cdot)$ to form features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. A contrastive loss function is used to minimise the distance between similar images and maximise the distance between dissimilar images.

Bootstrap Your Own Latent (BYOL) consists of two neural networks that interact with one another: an online and a target network [18]. The architecture receives input in the form of an image $\boldsymbol{x}$. The online network consists of three parts: an encoder layer $f(\cdot)$ which creates representations $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$, a projection head $g(\cdot)$ which creates features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$, and a predictor layer $q(\cdot)$ which creates predictions $\boldsymbol{p_i}$. The target network is like the online network without the prediction layer and different weights. The target network's weights are updated with the slow-moving average from the online network's weights (momentum encoding step). BYOL uses an L2 loss function to minimise the distance between similar input images. BYOL is more robust to different image augmentation processes than other contrastive learning architectures. Figure 3.2 shows the BYOL architecture.

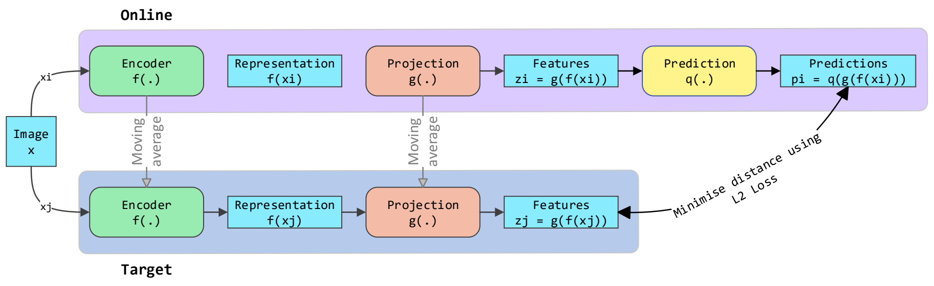

Figure 3.2: The BYOL architecture takes an input image $\boldsymbol{x}$ and creates two instances $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$. An encoder $f(\cdot)$ is used to create representation $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$. These representations pass through a projection head $g(\cdot)$ to form the features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. The online network passes the features through a prediction layer $q(\cdot)$ to create predictions $\boldsymbol{p_i}$. L2 loss is used to minimise the distance between the prediction $\boldsymbol{p_i}$ and the features $\boldsymbol{z_j}$.

Swapping Assignments between multiple Views of the same image (SwAV) is an online clustering-based self-supervised architecture [19]. It receives input in the form of an image $\boldsymbol{x}$ which is passed through an encoder $f(\cdot)$ to create representations $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$. These representations then pass through the projection head $g(\cdot)$ to create features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. Prototypes (C) are formed that are trainable vectors used to create clusters that each describe a characteristic of the data. Codes Qi and Qj are formed by assigning features to prototype vectors. Finally, a swapped prediction is made between the codes. SwAV uses online clustering to assign features to a prototype vector using the Sinkhorn Knopp algorithm. Sinkhorn Knopp is a transportation optimisation algorithm that minimises the difference between two probability distributions using iterative normalisation. This architecture also introduces a new image augmentation process known as multi-crop. Multi-crop creates multiple augmentations of the same image without quadratically increasing the computational effort required. The idea is to create two high-resolution images as well as multiple lower-resolution images. Figure 3.3 shows the SwAV architecture.

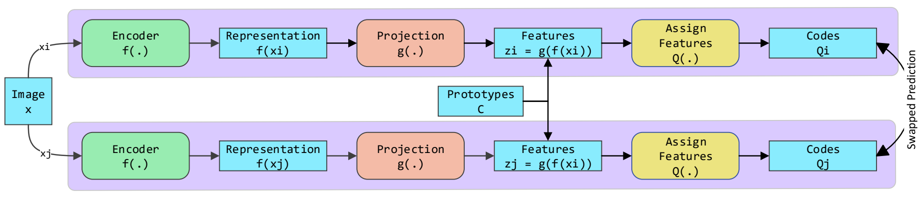

Figure 3.3: The SwAV architecture takes an input image $\boldsymbol{x}$ and creates two instances $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$. An encoder $f(\cdot)$ is used to create representation $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$. These representations then pass through a projection head $g(\cdot)$ to form the features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. Prototype vectors C are created and features are assigned to form codes Q_i and Q_j. These codes are used to perform swapped prediction to minimise the distance between similar images.

An architecture known as SimSiam (simple Siamese network), which is similar to SimCLR, is presented in [20]. This architecture takes an input image $\boldsymbol{x}$, creates a copy $\boldsymbol{x_i}$ and an augmentation $\boldsymbol{x_j}$, and passes it through an encoder $f(\cdot)$ to create representations $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$ which are then passed to a prediction layer such as an MLP layer $g(\cdot)$ to create features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. The prediction layer $q(\cdot)$ creates a prediction $\boldsymbol{p_i}$. It uses symmetric loss to minimise the negative cosine similarity between the prediction $\boldsymbol{p_i}$ and the feature $\boldsymbol{z_j}$. The paper shows that SimSiam can learn representations without the need for negative sample pairs, large batches, and momentum encoders. The authors of SimSiam describe SimSiam as:

> *"SimCLR without the negative pairs, SwAV without online clustering and BYOL without the momentum encoder."*

An important aspect of the SimSiam is its use of gradient stopping. Gradient stopping is applied to only one network, which leads to asymmetry in the Siamese network, which prevents the entire architecture from collapsing. Figure 3.4 shows the SimSiam architecture.

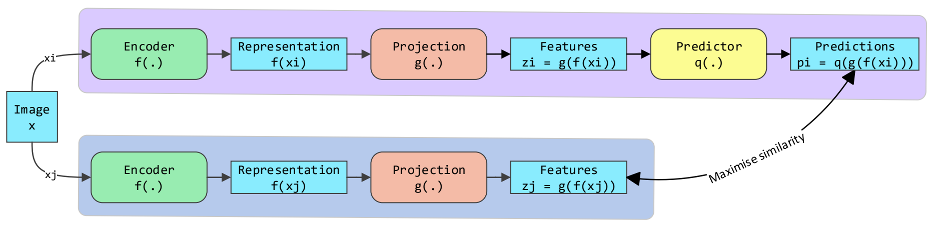

Figure 3.4: The SimSiam architecture takes an input image $\boldsymbol{x}$, and creates two instances $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$. An encoder $f(\cdot)$ is used to create representation $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$. These representations then pass through a projection head $g(\cdot)$ to form the features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. One network has a prediction layer $q(\cdot)$ which creates a prediction vector $\boldsymbol{p_i}$. The loss function is used to minimise the distance between the prediction vector $\boldsymbol{p_i}$ and the feature vector $\boldsymbol{z_j}$.

COntrastive Learning for Audio (COLA) is a general-purpose architecture used to learn audio representations [21]. The architecture receives a sampled audio signal as input data. From these signals, two augmentations are created by selecting different sections of the audio. These audio augmentations are then transformed into log-compressed mel-filterbank images $(\boldsymbol{x_i}, \boldsymbol{x_j})$ which serve as the input to the encoder model. The encoder $f(\cdot)$ maps the mel spectrogram images to a lower dimensional representation, followed by a projection head $g(\cdot)$ that maps the representations to the feature space. Instead of using cosine similarity to calculate the distance between the input images, this architecture uses bi-linear similarity. Bi-linear similarity is a generalisation of cosine similarity with different weights applied to certain features, which is appropriate in the context of audio processing where some components can be identified as noise. The loss function uses multi-class cross-entropy. The model is trained on segments of the same audio signal, using one segment as an anchor and the other segments as the positive samples of the same audio signal.

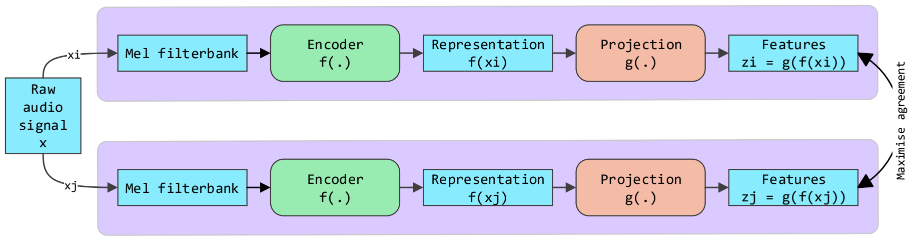

Figure 3.5: The COLA architecture takes a raw audio signal $\boldsymbol{x}$ as input. The raw audio signal is augmented to form two raw signals $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, which are then transformed into mel filterbank images. An encoder $f(\cdot)$ is used to create representations $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$ from these images. The representations then pass through a projection head $g(\cdot)$ to form features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$. The loss function uses a multi-class cross-entropy function to minimise the distance between similar images.

Contrastive Learning of Auditory Representations (CLAR) takes in audio samples as input data, similar to COLA [22]. However, in CLAR, one network is trained on the augmented audio signals (1D model) while the other network is trained on spectrograms of the augmented audio signals (2D model). Both networks pass the input data through an encoder $f(\cdot)$. These representations are then passed through a projection head $g(\cdot)$. The loss function that is a combination of cross-entropy L_{CE} and contrastive loss L_{CL}, $\therefore L_{CLAR} = L_{CE} + L_{CL}$. The authors considered six different image augmentation processes to determine which augmentations including pitch shift, noise injection, fade in/out, time masking, time shift, and time stretching. The researchers found that applying multiple augmentations to the same signal leads to the best results. For the 1D model, fade in/out, time stretching, and pitch shifting lead to the best results, compared to the 2D variant where fade in/out, time masking, and time shifting lead to the best results.

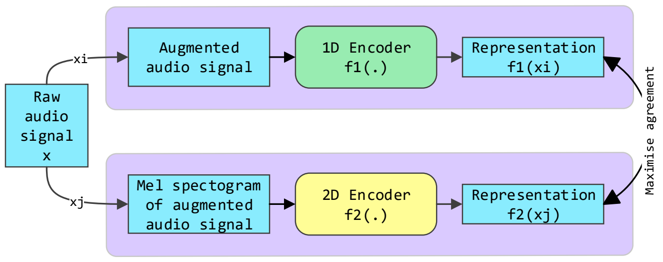

Figure 3.6: The CLAR architecture takes a raw audio signal $\boldsymbol{x}$ as input. The raw audio signal is augmented to form a 1D model with the augmented audio signal $\boldsymbol{x_i}$ and a 2D model with a mel spectrogram of the augmented audio signal $\boldsymbol{x_j}$. The 1D encoder $f_1(\cdot)$ is used with the augmented audio signal to create a representation $f(\boldsymbol{x_i})$ and 2D encoder $f_2(\cdot)$ is used with the augmented mel spectrogram to create a representation $f(\boldsymbol{x_j})$. The loss function consists of a combination of cross-entropy and contrastive loss and is used to maximise agreement.

3.3. Conclusion

This chapter focused on two sub-sections: firstly, the existing research in the field of TB classification, and secondly, the existing research in the field of contrastive learning. The existing research in the field of TB classification showed that there is a need for this project and that it is possible to classify TB patients using machine learning algorithms. The existing research in the field of contrastive learning explored the different architectures that can be used to achieve contrastive learning and showed that it can be applied to both image and audio classification problems.

Chapter 4

Data

This chapter will describe the datasets used for experimentation in later chapters. The datasets correspond to those that have been compiled and used by the DSP group in previous research, [4], [6], [5], [7].

4.1. Data sources

Two datasets were used for experimentation. Their names are based on the name of the clinic where they were recorded.

4.1.1. Brooklyn dataset

This dataset consists of coughs from patients who are TB-positive and their recent contacts. Thus it consists of coughs that are either from TB-positive patients or healthy patients. There are 38 patients in total, of which 17 are from known TB-positive patients and 21 are from healthy control patients. From these 38 patients, 746 cough samples were recorded, where 501 cough samples are from patients that are TB positive and 245 cough samples are from healthy patients. The coughs were recorded using a Rode M3 microphone and an audio field recorder at a sampling rate of 44.1 kHz [4].

	TB positive	**Healthy**	**Total**
Patients	17	21	38
Cough samples	501	245	746

Table 4.1: The Brooklyn dataset's setup consisting of TB-positive and healthy patients.

4.1.2. Wallacedene dataset

This dataset consists of coughs from patients who are TB-positive and coughs from patients who are TB-negative but have another pulmonary disease. There are 51 patients in total, of whom 35 are known to be TB negative and 16 TB positive. From these patients, 1358 cough samples were recorded, of which 956 are from patients that do not have TB and 402 are from patients that have TB. The coughs were recorded using a Rode M1 microphone and an audio field recorder at a sampling rate of 44.1 kHz [6].

	TB positive	**TB negative**	**Total**
Patients	16	35	51
Cough samples	402	956	1358

Table 4.2: The Wallacedene dataset's setup consisting of TB-positive and TB-negative patients.

Pre-processing

The raw recordings were stored in uncompressed pulse-code modulation (PCM) format. These recordings were then pre-processed by isolating the individual cough sounds and annotating the associated start and end times. The identified cough sounds were then divided into consecutive overlapping frames to which a Discrete Fourier Transform (DFT) was applied. Filterbanks were applied to these spectra, resulting in spectrograms with controlled frequency resolution. The dataset is stored as pickled versions [1] of linear filterbank spectrograms, mel filterbank spectrograms (mel spectrograms), and mel frequency cepstrum coefficient (MFCCs).

4.1.3. Combined dataset

Since the two datasets were very small, they were combined to contain cough samples from TB-positive patients and TB-negative patients, where the negative patients are either healthy or have a known pulmonary disease other than TB. The results are summarised in Table 4.3.

	TB positive	**TB negative**	**Total**
Patients	33	56	89
Cough samples	903	1201	2104

Table 4.3: A combined dataset consisting of the Wallacedene and Brooklyn datasets.

4.2. Cross validation

The data was split into twelfth folds using nested cross-validation, with three outer folds, each with four inner folds as shown in Figure 4.1.

[1]A pickled version of the mel spectrograms, refers to the process of serializing and deserializing a Python object into bytes.

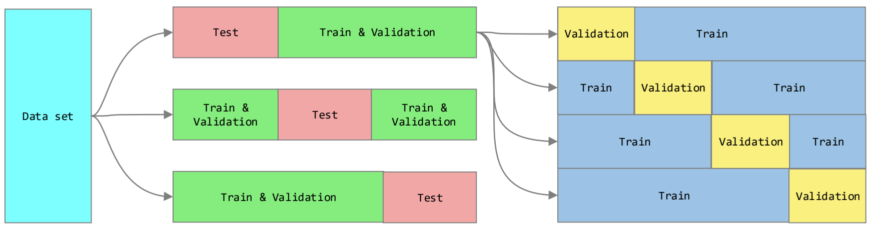

Figure 4.1: K-fold nested cross-validation process with three outer folds divided into four inner folds.

Table 4.4 shows how the combined dataset is split into three outer folds, each split into 4 inner folds. Each outer fold is split into a training & validation ($\sim 67\%$) dataset and a test ($\sim 33\%$) dataset. The training & validation dataset is further split into training ($\sim 75\%$) and validation ($\sim 25\%$) datasets.

Outer	Inner	Train	Validation	Test
0	0	36	13	25
	1	37	12	
	2	37	12	
	3	37	12	
1	0	36	13	25
	1	37	12	
	2	37	12	
	3	37	12	
2	0	37	13	24
	1	38	12	
	2	38	12	
	3	38	12	

Table 4.4: Training, validation, and test splits for the combined dataset. A third of the data is used for testing, and the remainder is used for training. Three-quarters of the training data is used for training and the remainder is used for validation.

Due to computational complexity, this project will not make use of nested-cross validation. Instead, this project will use the different folds to train 12 different models and evaluate all the models using the held-out test sets. Thus, for each outer fold, 4 models will be trained and optimised using the respective validation set. Then each of the four models will be evaluated using the held-out test set (outer fold). Hyper-parameter optimisation is performed using the average performance of the 12 models on the repsective validation datasets.

Chapter 5

Experimental approach

This chapter will describe the experimental approach. Firstly, this chapter describes the implementation of the transformation processes that will be used to create augmented images. Secondly, this chapter looks at the different loss function implementations used in this project. Thirdly, this chapter will discuss the model implementation. Lastly, this chapter will discuss the processes used to optimise the hyper-parameters for the transformations, loss functions, and model configuration.

5.1. Transformations

Following the research presented in [22], six different transformation processes were considered namely: time-shift, time mask, frequency mask, fade in/out, time stretch, and uniform frequency offset. Another transformation process, amplitude compression was later added to the list of transformations. For each transformation process, code snippets are attached to show how each transformation was implemented using the simplest and most effective implementation.

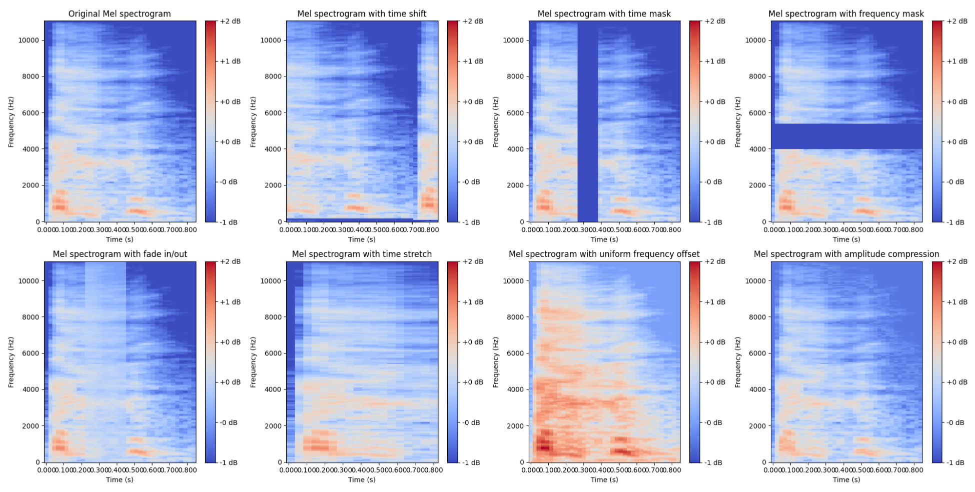

Figure 5.1: The first figure shows the original mel spectrogram that is created using 180 filters. The following figures show the effect of the different augmentation applied to the original mel spectrogram. The transformation processes shown are time-shifting, time masking, frequency masking, fade in/out, time stretching, uniform frequency offset, and amplitude compression.

Figure 5.1 shows the original mel spectrogram in the top left-hand corner, followed by the different augmented mel spectrograms. The mel spectrogram was computed using 37 time frames and 180 mel-spaced triangular filters in the filter bank.

5.1.1. Individual transformations

Time-shift

Time-shift was implemented by using Numpy's roll function. A random number is generated between 0 and the length of the mel spectrogram. That number is used as the number of indices to shift the array. In Figure 5.1, the second image shows the effect of time-shifting the original mel spectrogram, where the entire cough is shifted along the x-axis such that the start of the cough is now at 0.36s.

```python
def time_shift(mel):
    # mel is a two-dimensional array that contains the mel spectrogram,
   where each row is the spectrum of one frame.
    shift = np.random.randint(0, len(mel))
    return np.roll(mel, shift)
```

Time mask

Time masking occurs when the audio signal is suppressed over a certain time interval. To implement this, the width of the mel spectrogram was calculated. A random number between 1 and half the width plus 1 is generated for the width of the mask. Another random number is generated between 0 and the difference between the mel spectrogram's width and the mask's width to determine the start of the mask. The mask is then applied to the section between the mask start value and the mask start plus the mask width value. In Figure 5.1, the third image shows the effect of time masking the original mel spectrogram, where the section between 0.29s and 0.36s is masked to the minimum value of the mel spectrogram.

```python
def time_mask(mel):
    # mel is a two-dimensional array that contains the mel spectrogram,
   where each row is the spectrum of one frame.
    mel_width = mel.shape[1]
    mask_width = np.random.randint(1, mel_width/2 + 1)
    mask_start = np.random.randint(0, mel_width - mask_width + 1)
    mel[:, mask_start:mask_start + mask_width] = np.min(mel)
    return mel
```

Frequency mask

Frequency masking is implemented in a similar way to time masking. First, the height of the mel spectrogram is determined. A random number is generated between 1 and half of the height plus one is used to determine the mask's height. Another random number is generated between 0 and the difference between the mel spectrogram's height and the mask's height to determine the starting height of the mask. Finally, the mask is applied to the frequency section between the mask start and the mask start plus the mask height. In Figure 5.1, the fourth image shows the effect of frequency masking the original mel spectrogram, where a band of frequencies between 2000Hz and 3500Hz is masked to the minimum value of the mel spectrogram.

```
def frequency_mask(mel):
    # mel is a two-dimensional array that contains the mel spectrogram,
   where each row is the spectrum of one frame.
    mel_height = mel.shape[0]
    mask_height = np.random.randint(1, mel_height/2 + 1)
    mask_start = np.random.randint(0, mel_height - mask_height + 1)
    mel[mask_start:mask_start + mask_height, :] = np.min(mel)
    return mel
```

Fade in/out

The fade-in/out transform is similar to time masking. However, instead of setting the value to the minimum of the mel spectrogram as in time masking, the values are halved, creating a fading effect. In Figure 5.1, the fifth image shows the effect of fade in/out on the original mel spectrogram, where the section between 0.78s and 0.88s is slightly faded.

```
def fade(mel):
    # mel is a two-dimensional array that contains the mel spectrogram,
   where each row is the spectrum of one frame.
    mel_width = mel.shape[1]
    mask_width = np.random.randint(1, mel_width/2 + 1)
    mask_start = np.random.randint(0, mel_width - mask_width + 1)
    mel[:, mask_start:mask_start + mask_width] = 0.5*mel[:, mask_start:
   mask_start + mask_width]
    return mel
```

Time stretch

Time stretching is implemented by selecting half of the time interval of the mel spectrogram and then duplicating consecutive spectra. This is achieved by first creating a trimmed array containing the first half of all the columns in the original spectrogram. An augmented array is then created by repeating each column of the trimmed array. Lastly, a check is done to ensure that the dimensions of the original and augmented mel spectrograms match. If they do not match, then the augmented array is trimmed to the original size. In Figure 5.1, the sixth image shows the effect of time stretching on the original mel spectrogram, where each column is repeated twice, but the size of the mel spectrogram is still the same.

```
def time_stretch(mel):
    # mel is a two-dimensional array that contains the mel spectrogram,
   where each row is the spectrum of one frame.
    original_shape = mel.shape
    num_columns = int(mel.shape[-1]/2)
    trimmed_array = mel[..., :num_columns]
    repeat_factor = mel.shape[-1]
    aug_mel = np.repeat(trimmed_array, repeat_factor, axis=-1)

    if aug_mel.shape != original_shape:
        aug_mel = aug_mel[..., :original_shape[-1]]

    return aug_mel
```

Uniform frequency offset

Uniform frequency offset is applied by taking the transpose of the mel spectrogram, adding a constant value to the entire mel spectrogram, and taking the transpose again. In Figure 5.1, the seventh image shows the effect of uniform frequency offset the original mel spectrogram.

```
def uniform_frequency_offset(mel):
    # mel is a two-dimensional array that contains the mel spectrogram,
   where each row is the spectrum of one frame.
    mel = np.transpose(mel) + 0.5
    return np.transpose(mel)
```

Amplitude compression

Amplitude compression is applied by multiplying mel spectrogram values smaller than -0.75 and larger than 0.75, by 0.75 to place emphasis on the values in the range of -0.75 and 0.75. In Figure 5.1, the eighth image, shows the effect of compressing the amplitude of the original mel spectrogram.

```
def amp_compression(mel):
    # mel is a two-dimensional array that contains the mel spectrogram,
   where each row is the spectrum of one frame.
    mel[mel<-0.75] = mel[mel<-0.75]*0.75
    mel[mel> 0.75] = mel[mel> 0.75]*0.75
    return mel
```

5.1.2. Combinations

Four different combinations of the described annotated methods were tested to determine whether individual transformations or a combination of transformations deliver the best results. Combination 1 and combination 2 are derived using the results from [22]. Combination 3 is used to determine the effect of using all the time transformations and combination 4 is used to determine the effect of using time-shift and uniform frequency offset. Figure 5.2 shows the effect of the different combinations on the original mel spectrogram.

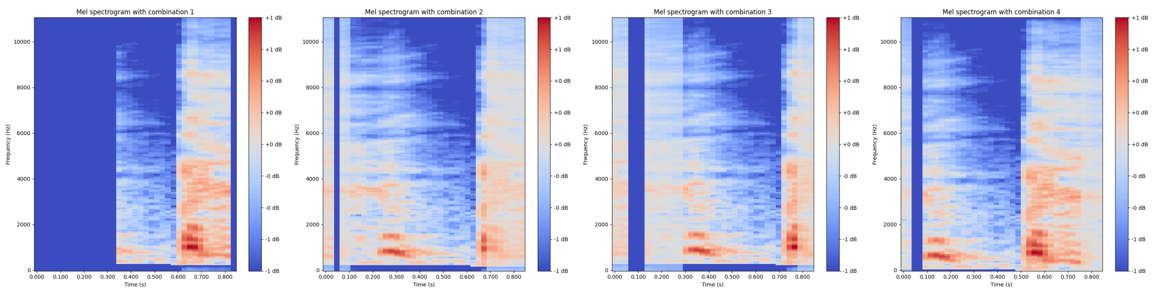

Figure 5.2: Each figure shows the effect of the different combinations applied to the original mel spectrogram. Combination 1 includes fade in/out, time masking and time-shifting. Combination 2 includes fade in/out, time stretching, and uniform frequency offset. Combination 3 includes time masking, time-shifting and time stretching. Combination 4 includes time-shifting and uniform frequency offset.

Combination 1 consists of fade in/out, time masking, and time-shifting. It is based on the findings presented in [22] for the 1D model. In the paper, the 1D model's transformations are applied to the audio signal directly, whereas the combination is directly applied to the mel spectrogram images of the audio signal. Combination 2 consists of fade in/out, time stretching, and uniform frequency offset. This combination is based on the findings presented in [22] for the 2D model. In the paper, the 2D model's transformations are directly applied to the mel spectrogram images. Combination 3 consists of time masking, time-shifting, and time stretching. It evaluates the effect of all the time-altering processes on the mel spectrogram. Combination 4 consists of time-shifting and uniform frequency offset. It evaluates the effect of perturbing the time and frequency of the mel spectrogram.

5.2. Loss functions

For the purposes of experimentation, this project considered the normalised temperature-scaled cross-entropy (NT-Xent) loss function and contrastive loss function.

5.2.1. Contrastive loss

Contrastive loss (2.2), calculate the distance between the original results and the transformed results. If the images are the same ($Y = 0$), i.e. same label, the distance is added to the cumulative loss. If the images are not the same ($Y = 1$), i.e. the labels are different, the difference between the margin and the distance is added to the cumulative loss. Thus, hen the loss function is minimised, the distance between similar images is minimised and the distance between dissimilar images is minimised (maximising the distance between dissimilar images.)

```
import torch as th

def contrastive_loss(results, results_aug, y_label):
    loss = 0
    distance = th.norm(results - results_aug, dim=1)
    margin = 0.1

    for j in range(distance.shape[0]):
        % if (y_label[j,0] == y_label[0,0]): loss += distance[j]**2
        else: loss += th.clamp(margin-distance[j], min=0)**2
    return loss
```

5.2.2. Normalised temperature-scaled cross-entropy loss

Normalised temperature-scaled cross-entropy (NT-Xent) loss (2.1), calculates the cosine similarity between the representations of the original and transformed image. The diagonal entries of the similarity matrix are set to minus infinity to ensure that the same entries ($k = i$) are not added to the loss. A target variable is created to pair the original and transformed representations.

```
def nt_xent_loss(results, results_aug, temperature):
    x = th.cat((results, results_aug), dim=0)

    sim = F.cosine_similarity(x[None, :, :], x[:, None, :], dim=-1)
    sim[th.eye(x.size(0)).bool()] = float("-inf")

    target = th.arange(x.size(0)).to(device)
    target[0::2] += 1
    target[1::2] -= 1

    loss = F.cross_entropy(sim / temperature, target, reduction="mean")
    return loss
```

5.3. Model

The model that will be evaluated consists of a Siamese network and a classifier. The Siamese network consists of two identical sub-networks where each sub-network consists of an encoder model (ResNet) and a projection head (consisting of three linear layers with ReLU activation functions between each linear layer). The classifiers used to compare results include logistic regression and multi-layer perceptron.

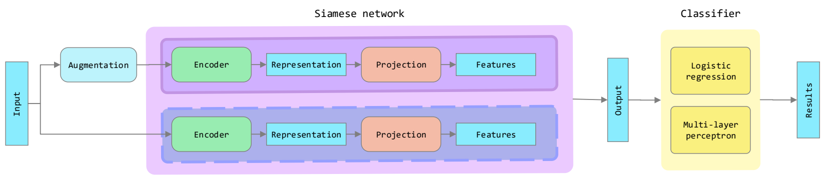

Figure 5.3: A flow diagram showing how the classifier is implemented. The model consists of a Siamese network followed by a classifier. The Siamese network is trained and optimised using the training and validation data. After the weights of the Siamese network has been optimised, its weights are frozen. The classifier is then trained using the training and test data, where the data is passed through the frozen Siamese network to create features that are used as input data for the classifier to test the entire model's performance.

The model is trained in three stages. The first stage is used to pre-train the sub-network. The second stage uses the pre-trained sub-network as the initial sub-network for the Siamese network (shown in purple), which is then trained on the training dataset, and validated on the validation set. Whilst the model is being trained, the weights of the one sub-network are updated during backpropagation, whilst the weights of the other sub-network remain unchanged. This strategy is used to prevent gradient collapse. The final stage uses the trained and optimised Siamese network to generate features for the original input images, which are then used to train the classifiers.

5.3.1. Siamese network

The Siamese network consists of two sub-networks, each of which consists of an encoder and a projection head. The weights of the sub-network that receives the augmented input are updated during backpropagation, whereas the weights of the other sub-network remain unchanged. This architecture is similar to the SimSiam architecture discussed in Chapter 3, where one sub-network has the prediction layer and the other sub-network has gradient stopping. Figure 5.4 shows how the Siamese network is implemented using a flow diagram.

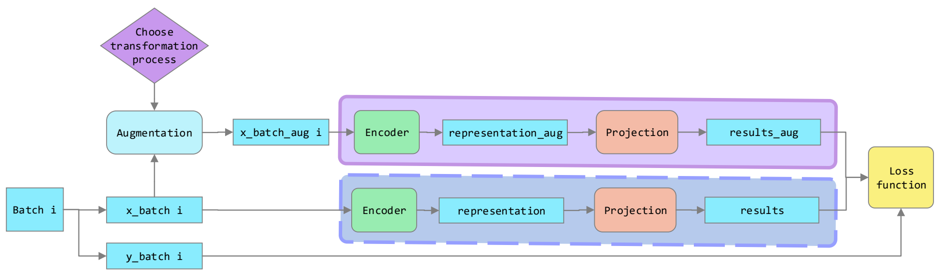

Figure 5.4: A flow diagram showing the training procedure for batch i. The batch is split into an *x_batch* and a *y_batch*. Using a selected transformation process and the *x_batch*, an augmented *x_batch_aug* is produced. The encoder is applied to each batch, followed by the projection head to produce *results* and *results_aug*. The loss function is applied to both results and the loss value is accumulated. The weights of the model with the augmented input (shown in purple) are updated during backpropagation, whilst the weight of the model with the normal input (shown in blue) remains unchanged.

The encoder model is applied to the original batch *x_batch* and the augmented batch *x_batch_aug* to produce representations of the input data. The representations are passed to the projection head which outputs the final results *results* and *results_aug*. The loss is calculated and accumulated over the batches. Finally, the weights of the sub-network with the augmented input are updated during backpropagation, whilst the weights of the sub-network with the normal input are not updated.

Figure 5.5 shows the implementation of one of the sub-networks in more depth. Each sub-network consists of an encoder such as a ResNet10, ResNet18, or ResNet34, and a projection head. The encoder model consists of a convolutional layer, a max pooling layer, four skip blocks, an average pooling layer, and a linear layer. The skip blocks each consist of 2 convolutional layers repeated twice, where a skip is implemented between each of the repeated layers. The projection head consists of three linear layers with ReLU layers in between each linear layer.

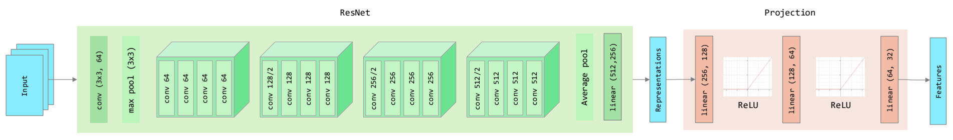

Figure 5.5: The sub-network consists of an encoder followed by a projection head. The sub-network shown uses a Resnet18 model as an encoder model, and the projection head consists of three linear layers with ReLU activation functions between each linear layer.

5.3.2. Classifiers

The classifiers are used to evaluate the performance of the approach as a whole. Once the Siamese network is trained and optimised, the weights are frozen. The classifiers are then trained using the training data and evaluated using the test data. The data is first passed through the Siamese network to generate the features, where the features are used as input for the classifier models.

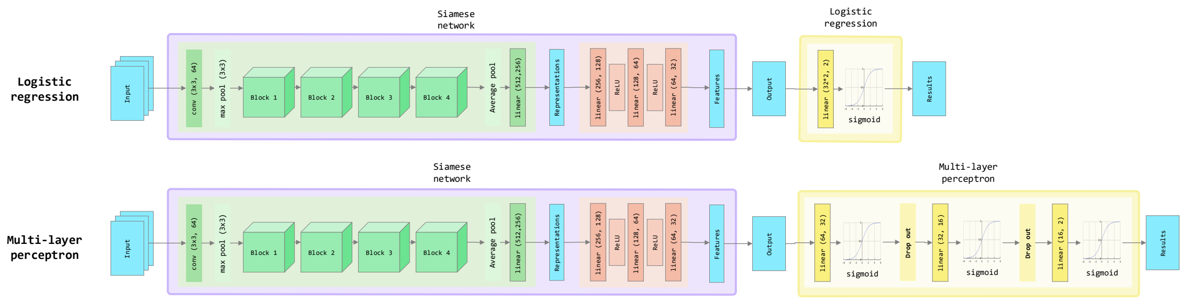

Figure 5.6: The complete classification system consists of a Siamese network that extracts features, followed by logistic regression or a multi-layer perceptron classifier.

Two classifiers are considered: logistic regression and a multi-layer perceptron as seen in Figure 5.6. Logistic regression is implemented as the baseline classifier since it is simple and often performs well. A multi-layer perceptron is implemented to test whether the features generated are more useful for a deeper classifier network. Logistic regression consists of a linear layer followed by a sigmoid layer using binary cross-entropy (BCE) loss as a loss function. The multi-layer perceptron consists of three linear layers with a sigmoid activation layer between each layer, followed by a sigmoid layer using binary cross-entropy (BCE) loss as a loss function.

5.4. Hyper-parameter optimisation

5.4.1. Selecting the transformation processes

The Siamese network model was trained on all the folds using all the processes discussed in Section 5.1. For each outer fold, the process that delivered the best average inner fold validation results was selected for mel spectrogram images calculated using 80 filters and mel spectrogram images calculated using 180 filters as input data. The same results were calculated first using a ResNet10 as the encoder model and then using a ResNet18 as the encoder model. Finally, this whole process was repeated first using NT-Xent loss, and then using contrastive loss. The psuedo-code for this process is provided in Appendix E, E.1. Tables C.1 and C.2 summarise the results of the process described above.

An initial subset of transformations was selected by considering all the transformations and combinations that appeared in Tables C.1 and C.2. This subset includes time-shift, uniform frequency offset, frequency mask, time mask, amplitude compression, combination 1, and combination 2.

For all the processes in the initial subset of transformations, the training loss was used to determine whether the models trained using that augmentation process converged. Of the initial subset, the training loss for the models using time-shifting, uniform frequency offset, and combination 2 converged. For the other transformations, the training loss was observed to either not converge or to diverge. Thus, the final subset of transformations included time-shifting, uniform frequency offset, and combination 2. Figure 5.7 shows the cumulative training loss calculated for each epoch using NT-Xent loss (left) and contrastive loss (right) for the final subset of transformation processes.

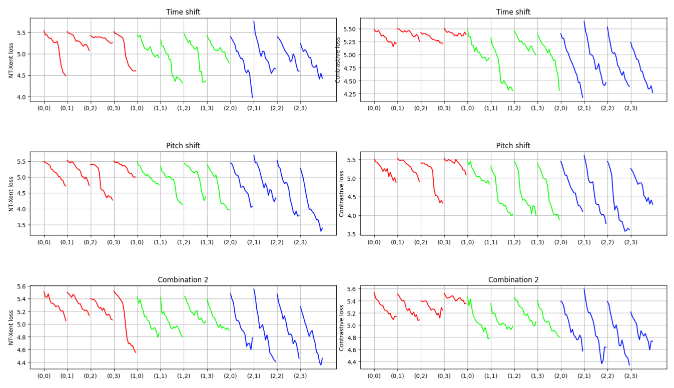

Figure 5.7: The training loss for NT-Xent and contrastive loss is shown for each of the selected augmentation processes. Each sub-figure shows the cumulative loss for a different augmentation process, where each color represents a different outer fold and each steep change indicates the start of a new inner fold.

5.4.2. Temperature selection

For the NT-Xent loss function, the temperature is a hyper-parameter that needs to be chosen. Three temperature values were considered: 0.1, 0.5, and 0.9. Using the same structure as defined above, three sets of experiments were conducted to determine which temperature value produces the best validation results for NT-Xent loss. The optimisation was only conducted using NT-Xent loss, using a ResNet10 as an encoder and mel spectrograms calculated with 80 filters. The psuedo-code for the process is provided in Appendix E, E.2.

For each outer fold, the highest average area under the curve for the development set was calculated for each of the transformation processes used to augment the images. Table 5.1 summarises the development results. From Table 5.1, the highest area under the curve for each outer fold is obtained for a temperature of 0.1. Therefore, a temperature of 0.1 will be used whenever the NT-Xent loss function is used.

Outer fold	Temperature	Process	Value
0	0.1	Time-shift	0.640
1	0.1	Combination 1	0.654
2	0.1	Uniform frequency offset	0.586

Table 5.1: The model's validation results for a ResNet10 encoder, using a mel spectrogram with 80 filters, at temperatures 0.1, 0.5, and 0.9.

5.4.3. Model configuration

Addition of projection layer

The initial Siamese model consisted of two sub-networks that each contained just an encoder model, such as a ResNet10 or ResNet18. Following the results presented in [16], an additional projection head (consisting of three linear layers with ReLU activation functions between each layer) was subsequently added to the sub-networks.

Preventing gradients from collapsing

One of the biggest problems encountered when training Siamese networks is gradient collapse, where the features map to the exact same embedding for the same input data. The authors of SimSiam proposed a simple technique known as gradient stopping to prevent gradient collapse [20]. Gradient stopping was implemented by updating the weights of one sub-network and keeping the weights of the other sub-network constant, where a pre-trained network is used for initial weights for both sub-networks.

Table 5.2 provides a summarised table of the validation-set results, showing the results for the model that consisted only of an encoder model, followed by the results obtained for the model that consisted of the encoder model and the projection head. Lastly, the results are shown using the encoder model and projection head with gradient stopping implemented. Complete sets of results are provided in Appendix C.

Model configuration	**Loss function**	**Result**
Encoder model only	NT-Xent	0.696
	Contrastive	0.674
Encoder model and projection head	NT-Xent	0.685
	Contrastive	0.711
Gradient stopping	NT-Xent	0.786
	Contrastive	0.813

Table 5.2: Validation results for sub-networks with only an encoder model and sub-networks with an encoder model and projection head. The results indicate that, for both loss functions, the highest area under the curve values are obtained when introducing the projection layer in conjunction with gradient stopping. Complete sets of validation-set results are provided in Appendix C.

The addition of the projection head shows an increase in the overall area under the curve values. This indicates that the model benefits from a deeper architecture. The addition of gradient stopping achieves a significant increase in performance for both loss functions. This indicates that gradient stopping plays a significant role in improving the overall model's performance on the validation set when the projection head is included.

5.5. Conclusion

This chapter describes the experimental approach that had been used during this project. The first section focused on the transformations that are used to create an augmentation of the original image. Seven transformations were implemented, along with four combinations of transformations. A subset of these transformations was chosen for use in the remainder of the project, including time-shifting, uniform frequency offset, and combination 2. The second section considered the implementations of the loss functions used to perform contrastive learning, including contrastive loss and normalised temperature-scaled cross-entropy loss. During hyper-parameter optimisation, a temperature of 0.1 was found to produce the best performance results on the validation dataset for NT-Xent loss. The third section dealt with the model implementation. It was shown, during hyper-parameter testing, that the Siamese network benefits from having a projection head after the encoder model, and that this benefit was maximised by adding gradient stopping during training.

Chapter 6

Experimental results

The previous chapter discussed the transformations used to create augmented images, loss function implementations, and the model's configuration consisting of a Siamese network and a classifier. This chapter will evaluate the performance of the Siamese network as a pre-task feature extractor by using the output from the Siamese network as the input to a classifier.

The Siamese network consists of an encoder model and a projection head, where the encoder model uses a ResNet34, and the projection model consists of three linear layers with ReLU activation functions between each layer. The output of the Siamese network is used as the input to two classifiers. In order to determine whether the Siamese network offers improved features, the same classifiers are also applied to the flattened input data. This allows a direct comparison between the performance results obtained when using the Siamese network as a feature extractor (where those features are used as input to the classifiers) and the results obtained without using the Siamese network (using the flattened input data as the input for the classifiers). Figure 6.1 shows a flow diagram of how this will be implemented.

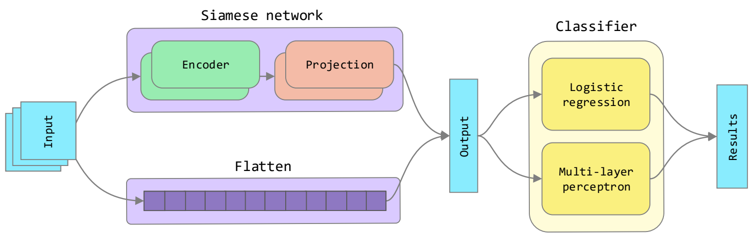

Figure 6.1: The flow diagram shows the testing setup. The top branch passes the input data to the Siamese network and then passes the features computed by the Siamese network to the classifiers. The bottom branch flattens the input data and passes this directly to the classifiers.

6.1. Test results

To evaluate the performance of the models with and without the Siamese network, the outer fold test sets were presented to each model. Because the test set and the training data are kept separate, one can test each model's performance on the respective test set without data leakage.

The trained and optimised Siamese network is used to generate features, where those features are used as input data for the classifiers. The performance results are calculated on the test set using the Siamese network (to generate features as input for the classifiers), and without using the Siamese network (using the flattened input as input for the classifiers). The process for evaluating the Siamese network is shown below in pseudo-code:

Algorithm 6.1: Pseudo code for the test setup

```
1: for classification results with Siamese network, and without Siamese network do
2:     for loss function ∈ {NT-Xent loss, Contrastive loss} do
3:         for process ∈ {time shift, uniform frequency offset, combination 2} do
4:             for outer fold j ∈ {0,1,2} do
5:                 for inner fold i ∈ {0,1,2,3} do
6:                     Train model using the training set
7:                     Evaluate model on the test set:
8:                 end for
9:                 $average\_outer\_fold\_j\_test\_auc = \frac{\sum_{i \in 0,1,2,3} inner\_fold\_i\_test\_auc}{4}$
10:                $best\_outer\_fold\_j\_test\_auc = max_{i \in 0,1,2,3}(inner\_fold\_i\_test\_auc)$
11:            end for
12:            $average\_test\_auc = \frac{\sum_{j \in 0,1,2} average\_outer\_fold\_j\_test\_auc}{3}$
13:            $best\_test\_auc = max_{j \in 0,1,2}(best\_outer\_fold\_j\_test\_auc)$
14:        end for
15:    end for
16: end for
```

Table 6.1 provides a summary of the test results. For each augmentation (using the specified loss function), the best AUC value over all the outer folds and the average AUC value for all the outer fold averages are shown, first using logistic regression as a classifier and then using a multi-layer perceptron as a classifier. The complete test set of results is provided in Appendix D, (D.14).

Model	Loss function	Augmentation	Logistic regression		Multi-layer perceptron	
			Best AUC	Average AUC	Best AUC	Average AUC
With Siamese network	NT-Xent	Time shift	0.757	0.553	0.567	0.453
		Uniform frequency offset	**0.838**	0.559	0.678	0.531
		Combination 2	0.707	0.529	**0.775**	0.522
	Contrastive	Time shift	0.713	0.506	0.678	0.484
		Uniform frequency offset	0.807	**0.585**	0.656	0.521
		Combination 2	0.721	0.474	0.679	0.483
Without Siamese network	N.A.	Time shift	0.686	0.523	0.622	0.523
		Uniform frequency offset	0.700	0.532	0.613	**0.532**
		Combination 2	0.686	0.510	0.678	0.510

Table 6.1: Summarised test set classification results with and without the contrastive-trained Siamese network. The best and average area under the curve value per outer fold is shown for logistic regression and a multi-layer perceptron. This is done for each augmentation process, namely time-shift, uniform frequency offset, and combination two. This process is repeated for both loss functions. The results are compared to the results obtained by classifying the flattened input data directly.

6.2. Discussion

Comparing the best AUC results to the average AUC results, one can see that there is a significant difference in performance - more than 25% difference between the top best AUC (0.838 for uniform frequency offset using NT-Xent loss) and top average AUC (0.585 for uniform frequency offset using contrastive loss). This indicates that the Siamese network performs well under certain circumstances, but is not consistent across all folds.

The augmentation process that delivered the best average performance was uniform frequency offset, for both logistic regression and the multi-layer perceptron. Uniform frequency offseting also delivered the best top AUC results when logistic regression was used for classification, whereas combination two delivered the best top results for the multi-layer perceptron. This indicates that uniform frequency offset is a successful transformation for training Siamese networks for TB cough classification.

Between the two classifiers, the results obtained using logistic regression outperform the results obtained using the multi-layer perceptron, for both the top and the average performance results, as well as the results obtained without using the Siamese network. For both classifiers, NT-Xent loss performs better than contrastive loss overall.

Overall, the top AUC results are obtained using a Siamese network, (0.838 for logistic regression and 0.775 using the multi-layer perceptron). For the average AUC results, the results obtained without the Siamese network are more consistent than those obtained using the Siamese network. For logistic regression, the top average results are obtained using the Siamese network with uniform frequency offset, but for the multi-layer perceptron, the top average results are obtained without using the Siamese network. This might indicate that the multi-layer perceptron is overfitting on the data.

The results obtained when using logistic regression as a classifier indicate that contrastive learning does lead to improved performance, while the variability of these improvements across the folds indicates that overtraining may be occurring and that a larger dataset might allow further performance improvements.

Chapter 7

Summary and Conclusion

7.1. Summary

The aim of this project was to determine whether contrastive learning can beneficially be applied to the field of audio classification for automatic TB classification. Different architectures and loss functions were considered in Chapters 2 and 3 as possible implementations to perform contrastive learning.

Audio recordings of coughs that had been gathered from TB-positive and TB-negative patients at clinics were available. The audio recordings were transformed into mel spectrograms calculated using 180 filters to use as input for the model. The data was divided three outer folds each with four inner folds per outer fold, as described in Chapter 4.

To perform contrastive learning, augmentations of the original mel spectrogram images were calculated and fed into the Siamese network. Seven different augmentations were considered and implemented, as well as four combinations of those augmentations. The Siamese network takes an image, creates an augmentation of the image, and passes both through sub-networks to create representations of the input data. One of two loss functions is then used to minimise the distance between the true image and the augmented image. The weights of one of the sub-networks are updated using backpropagation. The whole process is described in Chapter 5.

After the Siamese network has been trained and optimised, the features are used as input data which is passed to one of two classifiers. To determine the benefit of using the Siamese network, the same input data was flattened and passed directly to the same classifiers. The results of both sets of experiments are summarised and discussed in Chapter 6.

7.2. Main findings

7.2.1. Augmentation processes

Seven image augmentation processes and four combinations of these processes were considered for this project. The processes chosen were common audio transformation processes such as time shifting, uniform frequency offset, time masking, frequency masking, time stretching, fade in and out, and amplitude compression. The combinations were based on findings presented in the research literature.

To limit the computational time and complexity of this project, a subset of transformations was selected based on the training loss and development results. The final subset included time shifting, uniform frequency offset, and combination two (consisting of fade in/out, time stretching, and uniform frequency offset). Of the three transformations, uniform frequency offset delivered the best results. This indicates the model generalises well to frequency and time perturbations in the case of TB classification using mel spectrogram images.

7.2.2. Loss functions

Two loss functions were considered for this project, namely contrastive loss and NT-Xent loss. Contrastive loss is used to explicitly minimise the Euclidean distance between images with the same labels, whilst maximising the distance between images with dissimilar labels. NT-Xent loss uses a softmax function with temperature-scaled cosine similarity to minimise the distance between similar images and maximise the distance between dissimilar images.

During the course of this project, the results were computed for both loss functions to determine which one performs better. Both loss functions produce Siamsese models that achieved a performance increase compared to classification without the Siamese network. The test results using NT-Xent loss performed the best overall.

7.2.3. Architectures

Contrastive learning is a relatively new field for image and audio classification. One important consideration that had to be kept in mind was gradient collapse. Gradient collapse occurs when the gradients become negligible due to the loss function used to minimise the distance between similar images. Each architecture proposes a solution to prevent gradient collapse.

The first tested model implementation consisted of a SimCLR-like architecture with two identical sub-networks each with an encoder and a projection model. A later implementation tested the effect of adding gradient stopping to the Siamese network to prevent gradient collapse. This was done by allowing the weights of the sub-network with the augmented image to be updated, whilst the weights of the other sub-network remained unchanged. This implementation performed well on both the validation and test sets. This shows that the Siamese network benefits from gradient stopping to prevent gradient collapse.

7.2.4. Overall findings

The aim of this project was to identify whether contrastive learning could be applied to the field of TB classification. The results indicate that when a uniform frequency offset is used to augment the input images, the model becomes more robust to frequency perturbations. Both contrastive loss and NT-Xent loss can be used to perform contrastive learning, but NT-Xent loss delivered the best overall results. The results also indicate that a Siamese network with two sub-networks (each consisting of an encoder model and a projection model) benefits from gradient stopping to prevent gradient collapse. This shows that contrastive learning can be used to perform automatic TB classification.

7.3. Future work

7.3.1. Augmentations

Due to time constraints, this project only considered seven different augmentation processes and four combinations of these processes, of which a subset of two augmentation processes (time shifting and uniform frequency offset), and one combination (combination two) was selected. For future studies, other augmentation processes can be considered, such as frequency shifting, pitch shifting, speed perturbation, noise addition, and reverberation, to name a few. Hyper-parameter optimisation can also be applied to the augmentation processes to determine the ideal parameters.

7.3.2. Loss functions

This project focused on implementing two different contrastive learning loss functions, namely, contrastive loss and NT-Xent loss. Triplet loss was discussed in the Background chapter, however, this loss was never tested for the final models. Future studies could include implementing triplet loss and other contrastive loss functions. Hyper-parameter optimisation can also be applied to further optimise the loss function's parameters.

7.3.3. Architectures

Contrastive learning is implemented using different architectures. The first model's architecture focused on implementing an identical Siamese network similar to SimCLR and COLA. A later implementation focused on implementing gradients stopping to prevent gradients from collapsing, similar to SimSiam. Other architectures can be implemented to compare how the structure of Siamese networks influences the performance of different contrastive learning approaches.

7.3.4. Data

The input for this project was an existing audio dataset coded as mel spectrogram images. Specifically, the mel spectrogram images used for training, validation, and testing were generated using 180 filters. Other parameterisations can be considered, such as mel spectrograms calculated using 80 or 140 filters, or MFCCs. One of the biggest limitations of this project was the size of the data available. For future studies, a larger dataset can be considered.

Bibliography

[1] WHO, “Tuberculosis,” 2023. [Online]. Available: https://www.who.int/health-topics/tuberculosis#tab=tab_1

[2] Tuberculosis (tb) test. https://my.clevelandclinic.org/health/diagnostics/22751-tuberculosis-tb-test. Cleveland Clinic, Accessed: November 5, 2023.

[3] L. de Vos, E. Mazinyo, D. Bezuidenhout, N. Ngcelwane, D. Mandell, S. Schriger, J. Daniels, and A. Medina-Marino, “Reasons for missed opportunities to screen and test for TB in healthcare facilities,” *Public Health Action*, vol. 12, no. 4, pp. 171–173, Dec 2022.

[4] G. H. R. Botha, G. Theron, R. M. Warren, M. Klopper, K. Dheda, P. D. van Helden, and T. R. Niesler, “Detection of tuberculosis by automatic cough sound analysis,” *Physiological Measurement*, vol. 39, no. 4, p. 045005, 2018. [Online]. Available: https://doi.org/10.1088/1361-6579/aab6d0

[5] M. Pahar, M. Klopper, R. Warren, and T. Niesler, “COVID-19 cough classification using machine learning and global smartphone recordings,” *Computers in Biology and Medicine*, vol. 135, p. 104572, 2021.

[6] M. Pahar, M. Klopper, B. Reeve, R. Warren, G. Theron, and T. Niesler, “Automatic cough classification for tuberculosis screening in a real-world environment,” *Physiological Measurement*, vol. 42, no. 10, p. 105014, oct 2021. [Online]. Available: https://doi.org/10.1088%2F1361-6579%2Fac2fb8

[7] M. Pahar, M. Klopper, R. Warren, and T. Niesler, “COVID-19 detection in cough, breath and speech using deep transfer learning and bottleneck features,” *Computers in Biology and Medicine*, vol. 141, p. 105153, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482521009471

[8] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” *Computing Research Repository (CoRR)*, vol. abs/1310.4546, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[9] R. Nag, “A Comprehensive Guide to Siamese Neural Networks,” 2022. [Online]. Available: https://medium.com/@rinkinag24/a-comprehensive-guide-to-siamese-neural-networks-3358658c0513

[10] M. Rahman, “Sinkhorn Knopp: Unraveling Optimal Transport for Data Alignment,” 2022. [Online]. Available: https://rmoklesur.medium.com/sinkhorn-knopp-unraveling-optimal-transport-for-data-alignment-7fe87e26c903

[11] A. Khoeini, “Collapsing Solutions in Self-Supervised Learning,” 2022. [Online]. Available: https://arashk.medium.com/collapsing-solutions-in-self-supervised-learning-adfadede4739

[12] B. Williams, “Contrastive Loss Explained,” 2020. [Online]. Available: https://towardsdatascience.com/contrastive-loss-explaned-159f2d4a87ec

[13] G. D. Yellapu, G. Rudraraju, N. R. Sripada, B. Mamidgi, C. Jalukuru, P. Firmal, V. Yechuri, S. Varanasi, V. S. Peddireddi, D. M. Bhimarasetty, S. Kanisetti, N. Joshi, P. Mohapatra, and kiran Pamarthi, “Development and Clinical Validation of Swaasa AI Platform for screening and prioritization of Pulmonary TB,” *medRxiv*, 2022. [Online]. Available: https://www.medrxiv.org/content/early/2022/10/17/2022.09.19.22280114

[14] Showkatian, E., Salehi, M., Ghaffari, H., Reiazi, R., Sadighi, N., “Deep learning-based automatic detection of tuberculosis disease in chest X-ray images.” *Polish Journal of Radiology, 87*, 2022. [Online]. Available: https://doi.org/10.5114/pjr.2022.113435

[15] M. Noroozi and P. Favaro, “Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles,” *Computing Research Repository (CoRR)*, vol. abs/1603.09246, 2016. [Online]. Available: http://arxiv.org/abs/1603.09246

[16] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, “A Simple Framework for Contrastive Learning of Visual Representations,” *Computing Research Repository (CoRR)*, vol. abs/2002.05709, 2020. [Online]. Available: https://arxiv.org/abs/2002.05709

[17] ImageNet, “Download ImageNet Data,” 2020. [Online]. Available: https://www.image-net.org/download.php#:~:text=The%20most%20highly%2Dused%20subset,images%20and%20100%2C000%20test%20images.

[18] J. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. Á. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, “Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning,” *Computing Research Repository (CoRR)*, vol. abs/2006.07733, 2020. [Online]. Available: https://arxiv.org/abs/2006.07733

[19] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments,” *Computing Research Repository (CoRR)*, vol. abs/2006.09882, 2020. [Online]. Available: https://arxiv.org/abs/2006.09882

[20] X. Chen and K. He, "Exploring Simple Siamese Representation Learning," *Computing Research Repository (CoRR)*, vol. abs/2011.10566, 2020. [Online]. Available: https://arxiv.org/abs/2011.10566

[21] A. Saeed, D. Grangier, and N. Zeghidour, "Contrastive Learning of General-Purpose Audio Representations," *Computing Research Repository (CoRR)*, vol. abs/2010.10915, 2020. [Online]. Available: https://arxiv.org/abs/2010.10915

[22] H. Al-Tahan and Y. Mohsenzadeh, "CLAR: Contrastive Learning of Auditory Representations," *Computing Research Repository (CoRR)*, vol. abs/2010.09542, 2020. [Online]. Available: https://arxiv.org/abs/2010.09542

[23] P. Antoniadis, "An Introduction to Contrastive Learning," 2023. [Online]. Available: https://www.baeldung.com/cs/contrastive-learning

[24] Weng, L, "Contrastive Representation Learning," 2021. [Online]. Available: https://lilianweng.github.io/posts/2021-05-31-contrastive/

Appendix A

Project plan

Month	Day	Proposed work
June	12	Intial meeting with Prof T. Niesler to discuss skripsie options.
		Research topics, chose skripsie option, perform initial research on topic.
July	25	First in person group meeting with the DSP labs.
	31	Report: start with the Introduction and Literary review.
August	9	Upload signed GA document.
	11	Coding: start with data augmentation processes.
	25	Report: start on the Data section.
September	11	Coding: implement ResNets
	18	Report: start on Experimental Approach
	25	Coding: implement loss functions
October	2	Coding: implement Siamese network
	9	Coding: train, validate and test Siamese network
	16	Report: start with the Results chapter
	23	Report: start with the Conclusion chapter
	30	Report: finishing details
November	6	Report: hand in.
	11	Poster and video submission
	22	Open day presentations

Table A.1: Project plan

Appendix B

Graduate Attributes

	Graduate Attribute	Section	Application
GA 1	Problem Solving	Chapters 3, 5, 6	This project is based on a research problem in an under-defined, but known field of TB classification. This project will refine an existing problem and improve the current research on TB classification by introducing contrastive learning. This project looks at different architectures and algorithms, which requires an understanding of the processes followed as well as an understanding of what the results mean.
GA 2	Application of scientific and engineering knowledge	Chapters 2, 3	This project is a machine learning-based problem that requires an understanding of Mathematical Statistics, Computer Science, and Engineering principles and processes to model and solve this problem. It requires a high-level understanding of neural networks and a fundamental understanding of machine learning algorithms and programming, as well as a lot of research in the field of optimization as it uses state-of-the-art machine learning architectures that are often abstract and underdeveloped.
GA 3	Engineering Design	Chapter 5	To apply contrastive learning to TB classification, requires a step-by-step process consisting of developing, testing, and deploying each algorithm, followed by analysing the results and drawing conclusions. Additionally, each algorithm consists of procedural design steps that need to be followed in the correct order to achieve the desired goal.

Table B.1: Graduate Attributes 1,2 and 3.

	Graduate Attribute	**Section**	**Application**
GA 4	Investigations, experiments, and data analysis	Chapters 4, 5, 6, 7	This project is an investigation on whether contrastive learning can be used for TB classification. During the project, different architectures of contrastive learning methods as well as different algorithms for classification will be investigated. Each of the different architectures and algorithms needs to be tested by running multiple computational experiments on the datasets. After the experiments are completed, the results from each architecture and algorithm need to be evaluated to draw a final conclusion, for example using cross-validation.
GA 5	Engineering methods, skills, and tools, including Information Technology	Chapters 5	This project will be completed using Python libraries such as PyTorch, Numpy, and Matplotlib to develop and test the models created. The coding will be done in Visual Studio Code. This project uses the AGILE method for development, with weekly stand-ups. Other tools used for this project include Overleaf - an online latex editor for writing the report, GitHub for code version control, and WSL to uses Ubuntu on a Windows machine.
GA 6	Professional and technical communication	Entire report	The project includes a written report and an oral presentation. These demonstrate competence to communicate effectively, both orally and in writing.
GA 8	Individual work	Entire report	The student will take primary responsibility for the successful completion of all aspects of the project.
GA 9	Independent Learning Ability	Entire report	For successful completion of the project, the student is required to acquire knowledge independently (from the literature or the internet, for example) and without the context of this required knowledge being fully specified in the project definition.

Table B.2: Graduate Attributes 4,5,6,8 and 9.

Appendix C

Hyper-parameter optimisation

C.1. Encoder model

Outer fold	Filter size	ResNet	Process	Value
0	80	10	Combination 2	0.607143
1			Pitch shift	0.573661
2			Time shift	0.579464
0	180		Time mask	0.51875
1			Frequency mask	0.604911
2			Time shift	0.582143
0	80	18	Combination 1	0.625
1			Pitch shift	0.613839
2			Pitch shift	0.596429
0	180		Time shift	0.695536
1			Time shift	0.553571
2			Combination 1	0.549851

Table C.1: Area under the curve results for a ResNet10 and ResNet18 encoder, indicating the best transformation that produced the best results using a mel spectrogram with 80 and 180 filters, and using nt-xent loss as the loss function.

Outer fold	Filter size	ResNet	Process	Value
0	80	10	Combination 1	0.674107
1			Frequency shift	0.611607
2			Combination 1	0.643304
0	180		Combination 1	0.60625
1			Combination 2	0.671875
2			Combination 2	0.598214
0	80	18	Frequency shift	0.632143
1			Combination 1	0.584821
2			Time mask	0.567857
0	180		Time mask	0.657143
1			Frequency shift	0.647321
2			Amplitude compression	0.669048

Table C.2: Area under the curve results for a ResNet10 and ResNet18 encoder, indicating the best transformation that produced the best results using a mel spectrogram with 80 and 180 filters, and using the contrastive loss function.

C.2. Encoder and projection model

Folder	Filter size	Resnet	Process	Value
0	80	10	Amp compression	0.59375
1			Combination 1	**0.685268**
2			Combination 2	0.67619
0	180		Combination 2	0.595536
1			Combination 1	0.587054
2			Combination 2	0.66503
0	80	18	Time mask	0.657143
1			Combination 2	0.622768
2			Amplitude compression	0.64628
0	180		Amplitude compression	0.614286
1			Combination 2	0.674107
2			Time shift	0.600149

Table C.3: Area under curve results for a ResNet10 and ResNet18 encoder, using a mel spectrogram with 80 and 180 filters using nt-xent loss.

Outer fold	**Filter size**	**ResNet**	**Process**	**Value**
0	80	10	Frequency mask	0.666071
1			Amplitude compression	0.609375
2			Combination 1	0.595685
0	180		Combination 2	0.708036
1			Frequency mask	0.658482
2			Frequency mask	0.586607
0	80	18	Combination 2	0.60625
1			Time mask	0.658482
2			Combination 1	0.677232
0	180		Frequency mask	0.636607
1			Combo 1	0.694196
2			Time mask	**0.710714**

Table C.4: Area under curve results for a ResNet10 and ResNet18 encoder, using a mel spectrogram with 80 and 180 filters using contrastive loss.

Appendix D

Results

D.1. Validation

D.1.1. Logistic regression

Augmentation	**Outer fold**	**Inner fold**	**Area under the curve**
Time shift	0	1	0.786
	1	1	0.607
	2	3	0.667
Frequency shift	0	2	0.536
	1	3	0.571
	2	0	0.563
Combination 2	0	3	0.661
	1	1	0.607
	2	1	0.600

Table D.1: Validation results using NT-Xent loss and logistic regression.

Augmentation	**Outer fold**	**Inner fold**	**Area under the curve**
Time shift	0	2	0.786
	1	2	0.679
	2	1	0.657
Frequency shift	0	2	0.661
	1	0	0.813
	2	1	0.514
Combination 2	0	2	0.679
	1	0	0.625
	2	1	0.429

Table D.2: Validation results using contrastive loss and logistic regression.

D.1.2. Multi-layer perceptron

Augmentation	Outer fold	Inner fold	Area under the curve
Time shift	0	1	0.464
	1	2	0.732
	2	0	0.688
Frequency shift	0	0	0.686
	1	3	0.589
	2	3	0.700
Combination 2	0	1	0.786
	1	0	0.625
	2	0	0.563

Table D.3: Validation results using NT-Xent loss and a multi-layer perceptron.

Augmentation	Outer fold	Inner fold	Area under the curve
Time shift	0	1	0.732
	1	2	0.732
	2	3	0.717
Frequency shift	0	1	0.686
	1	2	0.679
	2	3	0.786
Combination 2	0	1	0.732
	1	3	0.518
	2	3	0.633

Table D.4: Validation results using contrastive loss and a multi-layer percepton.

D.1.3. Normal model results

Augmentation	Outer fold	Inner fold	Area under the curve
Time shift	0	3	0.786
	1	1	0.589
	2	2	0.857
Frequency shift	0	3	0.714
	1	1	0.589
	2	2	0.786
Combination 2	0	3	0.786
	1	1	0.589
	2	2	0.857

Table D.5: Validation results for a logistic regression model with flattened input.

Augmentation	Outer fold	Inner fold	Area under the curve
Time shift	0	2	0.661
	1	1	0.661
	2	0	0.813
Frequency shift	0	1	0.607
	1	2	0.607
	2	1	0.586
Combination 2	0	2	0.571
	1	0	0.500
	2	1	0.657

Table D.6: Validation results for a multi-layer perceptron with flattened input.

D.2. Test

D.2.1. Logistic regression

Augmentation	**Outer fold**	**Inner fold**	**Area under the curve**
Time shift	0	3	0.700
	1	2	0.629
	2	1	0.613
Frequency shift	0	1	0.478
	1	2	0.600
	2	1	0.675
Combination 2	0	2	0.589
	1	2	0.536
	2	2	0.708

Table D.7: Test results using NT-Xent loss and logistic regression.

Augmentation	**Outer fold**	**Inner fold**	**Area under the curve**
Time shift	0	2	0.533
	1	2	0.636
	2	1	0.708
Frequency shift	0	2	0.611
	1	1	0.607
	2	1	0.742
Combination 2	0	2	0.600
	1	1	0.636
	2	0	0.550

Table D.8: Test results using contrastive loss and logistic regression.

D.2.2. Multi-layer perceptron

Augmentation	**Outer fold**	**Inner fold**	**Area under the curve**
Time shift	0	0	0.644
	1	0	0.500
	2	3	0.542
Frequency shift	0	2	0.600
	1	0	0.650
	2	1	0.608
Combination 2	0	2	0.756
	1	3	0.586
	2	1	0.579

Table D.9: Test results using NT-Xent loss and a multi-layer perceptron.

Augmentation	**Outer fold**	**Inner fold**	**Area under the curve**
Time shift	0	1	0.732
	1	2	0.732
	2	3	0.717
Frequency shift	0	0	0.686
	1	2	0.679
	2	2	0.786
Combination 2	0	1	0.732
	1	3	0.518
	2	3	0.633

Table D.10: Test results using contrastive loss and a multi-layer percepton.

D.2.3. Normal model results

Augmentation	Outer fold	Inner fold	Area under the curve
Time shift	0	2	0.656
	1	1	0.686
	2	1	0.579
Frequency shift	0	0	0.633
	1	0	0.629
	2	2	0.671
Combination 2	0	0	0.678
	1	1	0.686
	2	1	0.579

Table D.11: Test results for a logistic regression model with flattened input.

Augmentation	Outer fold	Inner fold	Area under the curve
Time shift	0	1	0.622
	1	1	0.621
	2	1	0.608
Frequency shift	0	2	0.578
	1	0	0.536
	2	3	0.613
Combination 2	0	3	0.678
	1	0	0.607
	2	1	0.583

Table D.12: Test results for a multi-layer perceptron with flattened input.

D.3. Complete validation results

Model	Loss function	Augmentation	Outer fold	Logistic regression	Multi-layer perceptron
Siamese network	NT-Xent	Time shift	0	0.786	0.464
			1	0.607	0.732
			2	0.667	0.688
		Frequency shift	0	0.536	0.686
			1	0.571	0.589
			2	0.563	0.700
		Combination 2	0	0.661	0.786
			1	0.607	0.625
			2	0.600	0.563
	Contrastive	Time shift	0	0.786	0.732
			1	0.679	0.732
			2	0.657	0.717
		Frequency shift	0	0.661	0.686
			1	0.813	0.679
			2	0.514	0.786
		Combination 2	0	0.679	0.732
			1	0.625	0.518
			2	0.429	0.633
Flattened input	N.A.	Time shift	0	0.786	0.661
			1	0.589	0.661
			2	0.857	0.813
		Frequency shift	0	0.714	0.607
			1	0.589	0.607
			2	0.786	0.586
		Combination 2	0	0.786	0.571
			1	0.589	0.500
			2	0.857	0.657

Table D.13: Summarised validation results for the entire model. The highest area under the curve value per outer fold is selected using logistic regression and a multi-layer perceptron. This is done for each augmentation process, namely time shifting, frequency shifting and for combination two. This process is repeated for both loss functions.

D.4. Complete test results

Model	Loss function	Augmentation	Outer fold	Logistic regression		Multi-layer perceptron	
				Best AUC	Average AUC	Best AUC	Average AUC
With Siamese network	NT-Xent	Time shift	0	0.589	0.511	0.567	0.503
			1	0.757	0.588	0.514	0.457
			2	0.613	0.559	0.483	0.398
		Frequency shift	0	0.611	0.531	0.678	**0.575**
			1	0.636	0.443	0.600	0.532
			2	**0.838**	**0.703**	0.521	0.485
		Combination 2	0	0.500	0.467	0.589	0.522
			1	0.707	0.550	0.707	0.486
			2	0.671	0.571	**0.775**	0.510
	Contrastive	Time shift	0	0.567	0.503	0.678	0.500
				0.614	0.505	0.643	0.525
				0.713	0.510	0.517	0.428
		Frequency shift	0	0.578	0.497	0.656	**0.631**
				0.807	**0.666**	0.600	0.461
				0.738	0.593	0.646	0.471
		Combination 2	0	0.489	0.442	0.633	0.439
			1	0.721	0.450	0.643	0.525
			2	0.671	0.530	**0.679**	0.486
Without Siamese network	N.A.	Time shift	0	0.656	0.625	0.622	0.558
				0.686	0.593	0.621	0.570
				0.579	0.554	0.608	0.442
		Frequency shift	0	**0.700**	**0.639**	0.578	0.564
				0.621	0.573	0.536	0.504
				0.608	0.514	0.613	0.529
		Combination 2	0	0.678	0.633	**0.678**	**0.586**
			1	0.686	0.593	0.607	0.475
			2	0.579	0.546	0.583	0.470

Table D.14: Test results for the entire model. The highest and average area under the curve value per outer fold is selected using logistic regression and a multi-layer perceptron, for each augmentation process, for both loss functions. The results are calculated with contrastive learning and without contrastive learning.

The table shows the best (highest) AUC and the average AUC (calculated over all four inner folds) for each outer fold. This is shown for each augmentation process that is part of the final subset of transformations selected in Chapter 5, using both NT-Xent loss and contrastive loss. The results are computed first using logistic regression and then using a multi-layer perceptron to classify the results. The last few rows show the results obtained without using the Siamese network (using the flattened input data as input for the classifiers).

Appendix E

Pseudo code for algorithms

E.1. Transformation selection

Algorithm 5.2: Pseudo code for the selecting best transformation processes

```
1: for loss function ∈ {NT-Xent loss, Contrastive loss} do
2:     for resnet ∈ {ResNet10, ResNet18} do
3:         for filter size ∈ {80, 180} do
4:             for outer fold ∈ {0,1,2} do
5:                 for process ∈ {all transformation, all combinations} do
6:                     for inner fold ∈ {0,1,2,3} do
7:                         Train Siamese network using the training set
8:                         Evaluate Siamese network on the validation set
9:                     end for
10:                    Calculate average outer fold AUC
11:                end for
12:                Select process with the highest outer fold average AUC value
13:            end for
14:        end for
15:    end for
16: end for
```

E.2. Temperature selection

Algorithm 5.3: Pseudo code for the selecting best temperature for NT-Xent loss

```
 1: for temperature ∈ {0.1, 0.5, 0.9} do
 2:     for outer fold ∈ {0,1,2} do
 3:         for process ∈ {all transformation, all combinations} do
 4:             for inner fold ∈ {0,1,2,3} do
 5:                 Train Siamese network using the training set
 6:                 Evaluate Siamese network on the validation set
 7:             end for
 8:             Calculate average outer fold AUC
 9:         end for
10:         Select process with the highest outer fold average AUC value
11:     end for
12: end for
```

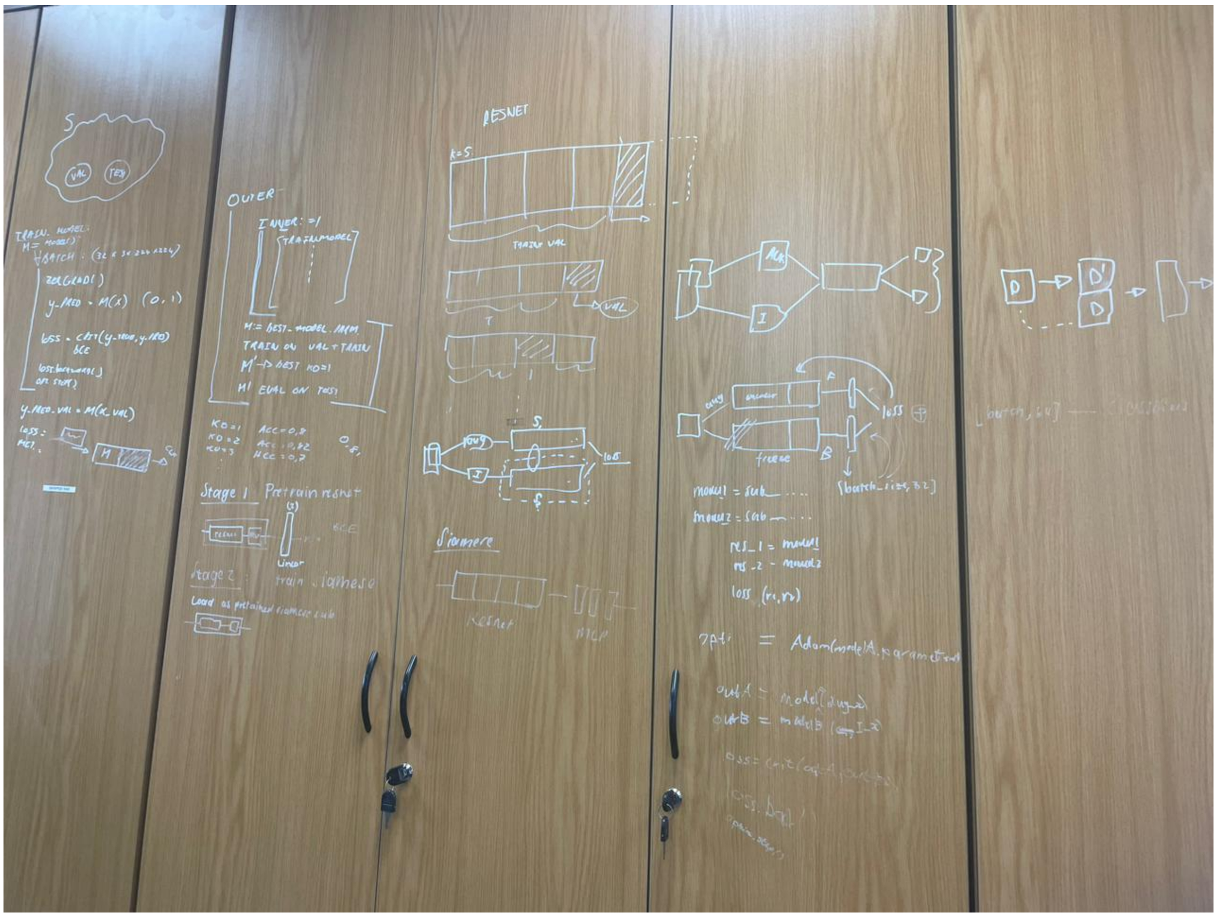

Figure E.1: DSP lab