# Chapter 5

# Experimental approach

This chapter will describe the approach used to perform the experiments for this project. The first section will focus on the training process that is used throughout the project. The next section will discuss the machine learning models and architectures. Followed by a preliminary results section.

## 5.1. Training

The model used for experiments for this project consists of a Siamese Network and a classifier, where the Siamese network is used to perform contrastive learning and the classifier is used to evaluate the performance of the contrastive learning approach relative to the baseline. The model is trained and tested using audio cough recordings. From these recordings, acoustic features (*not sure which word I should use*) such as spectrograms and MFCCs are calculated. These acoustic features are used as input data to the model. The input data is divided into training, validation and testing data using nested cross-validation, which will be discussed in more detail in section 5.1.1. The Siamese network is trained using the training data. The trained Siamese network is used with the classifier to evaluate the effect of different sets of hyper-parameters using the validation dataset. The set of hyper-parameters that deliver the best results on the validation data is the optimal set. After the model has been trained and optimised, the model is tested using the test data.

### 5.1.1. Nested cross validation

As mentioned above, this project makes use of nested cross-validation. During cross-validation, the dataset is split into equal smaller subsets. Each subset is referred to as an outer fold. For each training loop $i$, the model is trained using all of the outer folds except $outer\_fold_i$ which is used to test the model.

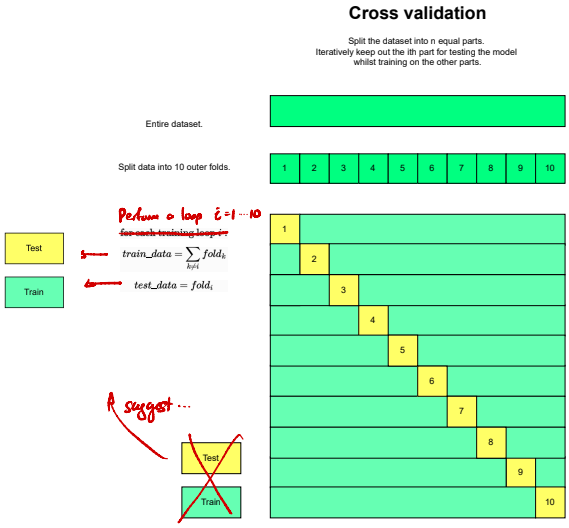


**Figure 5.1:** Cross-validation.

Nested cross-validation, is cross-validation within a cross-validation loop. The entire dataset is first split into n (in this case 10) outer folds. For each outer training loop $i$, the model is tested using the corresponding outer fold $test\_data = outer\_fold_i$, whilst the model is trained and validated using the remaining outer folds $outer\_training\_data = \sum_{j \neq i} outer\_fold_j$. For each outer fold, the model's hyper-parameters are optimised using inner loops. For each inner training loop $j$, where $j \neq i$, the model is evaluated on the corresponding inner fold $validation\_data = inner\_fold\_j$, whilst the model is trained on the remaining inner folds $inner\_training\_data = \sum_{k \neq i,j} inner\_fold_k$.

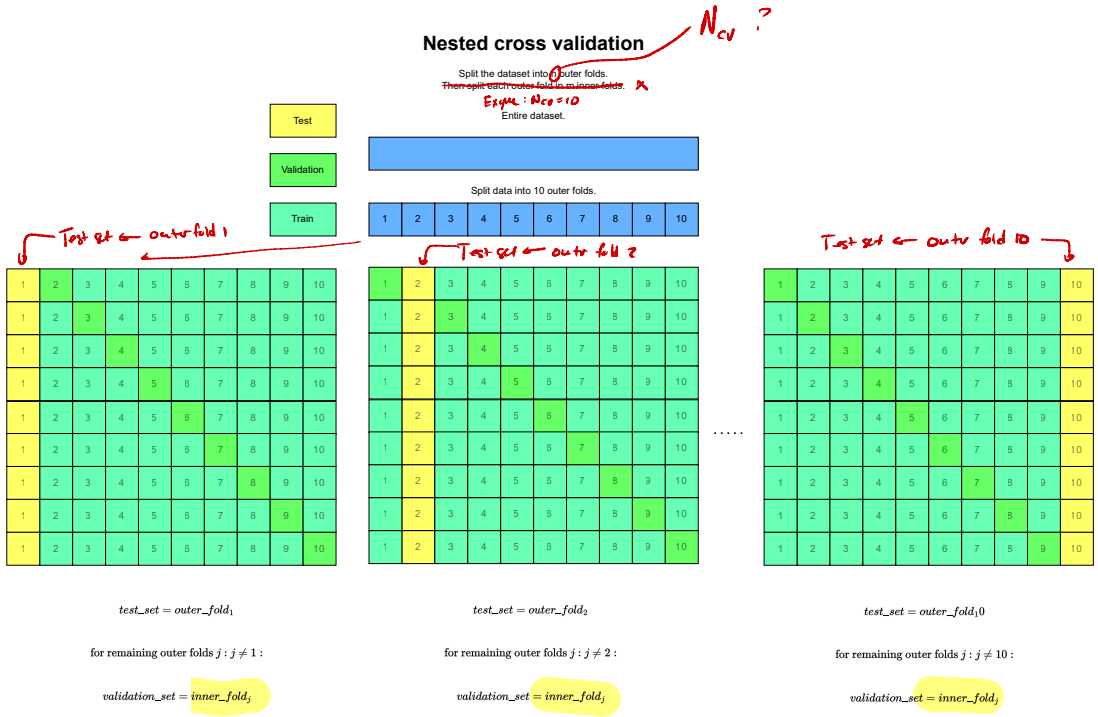


**Figure 5.2:** Nested cross-validation.

## 5.2. Logistic regression

A logistic regression model was implemented to use as a baseline classifier. The model consists of a linear layer that maps the input data to the output dimension. The model receives images as input data, which is first flattened and then passed to the linear layer. For this project, a 2 class multi-nomial logistic regression was implemented. The output corresponds to the likelihood that the input data is a TB-negative (0) or TB-positive (1) patient.

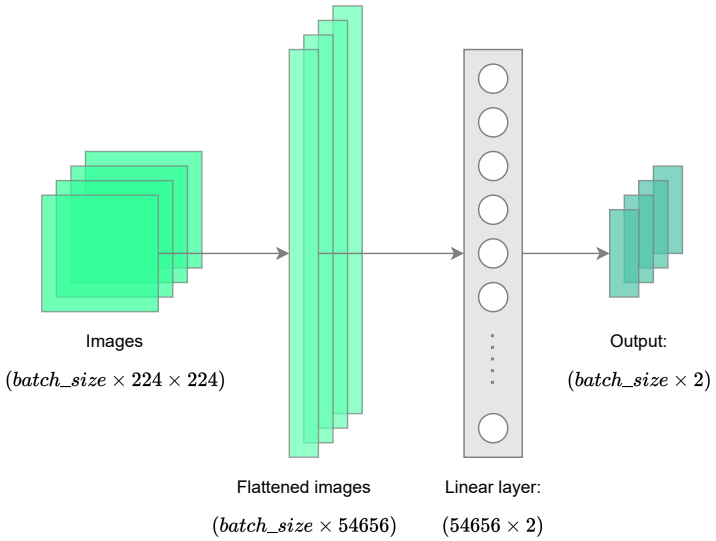


**Figure 5.3:** Each image in the batch is flattened and passed to the linear layer to generate 2 outputs for two-class classification.

The logistic regression model is also used later to compare the performance of the different Siamese architectures, where the input data is the features generated by the Siamese network instead of the flattened input images. For example, if the dimension of the features is $(batch\_size \times 128)$, then the linear layer will be $(128, 2)$ and the resulting output will still be $(batch\_size \times 2)$.

## 5.3. SimCLR architecture

The first Siamese architecture investigated for implementing contrastive learning combines the SimCLR and COLA architectures. The architecture is a Siamese model with two sub-networks each consisting of an encoder model and a projection head. The encoder is a standard ResNet model (different ResNet models were considered such as a ResNet18, ResNet34 and a ResNet50 based on the results published in the literature). The projection head consists of a fully connected linear layer $(256 \times 256)$, a batch normalisation layer, a ReLU layer, and another fully connected linear layer $(256 \times 128)$.

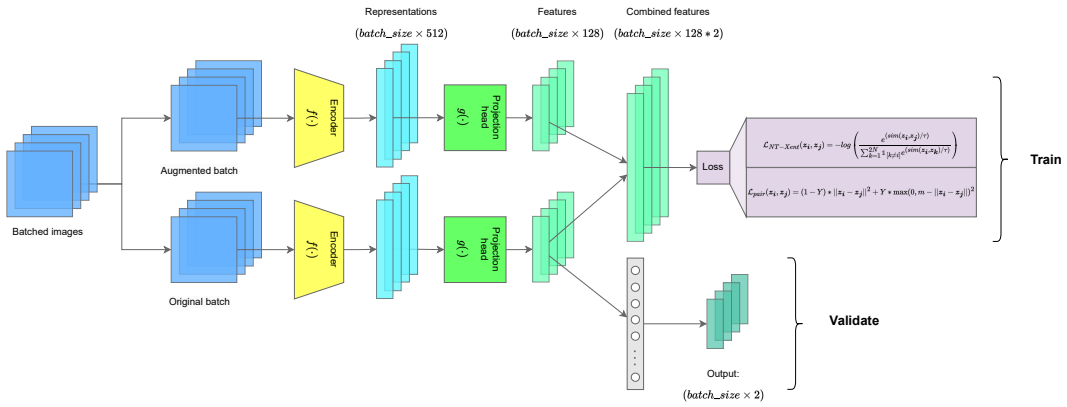


**Figure 5.4:** Implemented SimCLR architecture.

The Siamese network receives audio signals as input, ~~which are converted into~~ spectrograms using linear filter banks or mel-spectrograms using mel-scaled filter banks. The spectrogram images are divided into batches using a personalised DataLoader class. For each batch of images with shape $(batch\_size \times 224 \times 224)$, an augmented batch is created using one of the augmentation processes. The original (anchor) batch is passed to one sub-network of the Siamese network, while the augmented batch is passed to the other sub-network. The encoder models generate representations of shape $(batch\_size \times 512)$ of the original and augmented batch. These representations are then passed to the projection head to generate features of shape $(batch\_size \times 128)$ of the input images. These features are combined and passed to the loss function. For this architecture, ~~multiple~~ loss functions can be considered, for example, pair loss or normalised temperature-scaled cross-entropy loss. During back-propagation, the weights of the model are updated using the derivative of the loss with respect to the weights. To evaluate the performance of the architecture, these features are passed to the logistic regression model to generate the output of shape $(batch\_size, 2)$.

## 5.4. SimSiam architecture

The SimSiam architecture is similar to the SimCLR and COLA architecture above, with the addition of a prediction head. The sub-networks each consist of the encoder and the projection head followed by a prediction layer. The prediction layer is a simple multi-layer perceptron that consists of a linear layer, batch normalisation, a ReLU activation function and another linear layer, where the resulting predictions are similar in dimensions to the features. The loss function that is used with this architecture is a symmetric loss function, ~~as shown in~~ Equation 2.14. The distance is calculated between the prediction of the one sub-network ($\boldsymbol{p_i}$) and the detached features of the other sub-network ($\boldsymbol{z_j}$). Detaching the features ensures that the model does not collapse. Similar to the architecture above, the weights of the model are updated during back-propagation using the symmetric loss and

the performance of the architecture is evaluated by passing the predictions to the logistic regression model.

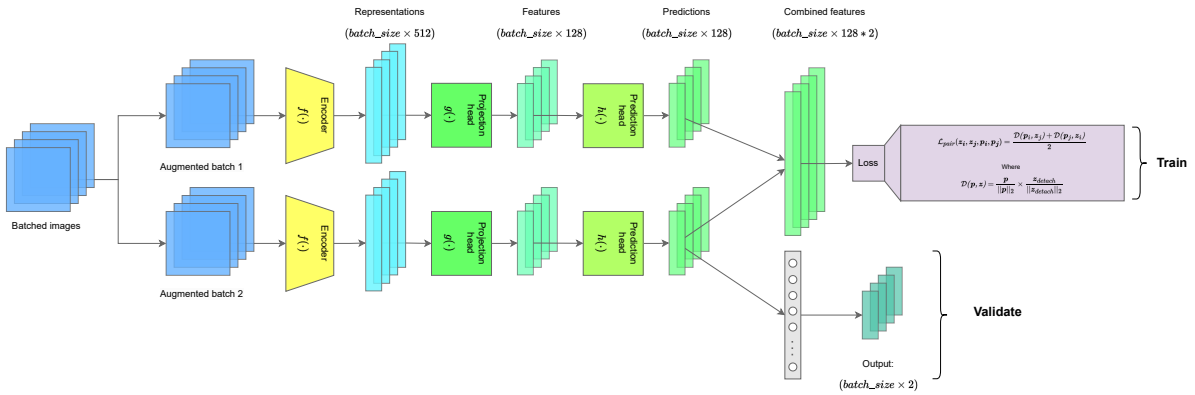


**Figure 5.5:** Implemented SimSiam

## 5.5. Hyper-parameter testing on Hyfe Data

The Hyfe audio samples are 0.5s long. The audio is padded with zeros at the end such that each audio signal contains $224 * 224$ samples. A design choice was made to calculate the overlap based on the required shapes for ResNet models. The parameters needed to generate 224 time steps are calculated as shown in Equation 5.1.

$$\begin{aligned} step\_size &= audio\_length/target\_width \\ overlapping\_samples &= window\_size - step\_size + 1 \\ overlap &= overlapping\_samples/window\_size \end{aligned} \tag{5.1}$$

In Equation 5.1, the step size between each window is calculated based on the desired target width. The overlap is calculated as the number of overlapping samples divided by the window size. For example, if $audio.shape = (1, 50176), target\_width = 224, window\_size = 256$ then

$$\begin{aligned} step\_size &= 50176/224 = 224 \\ overlapping\_samples &= 256 - 224 + 1 = 33 \\ overlap &= 33/256 = 0.129 \end{aligned}$$

The spectrograms are calculated using 0.129 overlap between samples and 447 FFTs. The target height is obtained by taking two times the desired height minus one FFTs and using only the first half of those FFTs to prevent aliasing as per the Nyquist Theorem (not sure about the wording). The spectrograms are calculated using 224 linearly spaced bins, whereas the mel-spectrograms are calculated using *num_mel_bins* mel-spaced bins. Finally, the MFCCs are calculated by applying a DCT to the mel-spectrograms and keeping *num_cepstrums* of the cepstral coefficients. For example, for 80 mel-spaced bins and 39 cepstral coefficients, the input data is transformed into images with the following

shapes:

| | |
|---|---|
| *Spectrograms* | $(224 \times 224)$ |
| *Melspectrograms* | $(224 \times 80)$ |
| *MFCCs* | $(224 \times 39)$ |

A traditional ResNet model expects the data's input shape to be $(224 \times 224)$. The spectrograms are already in the correct shape by ~~setting~~ the parameters as calculated above. However, the mel-spectrograms' and the MFCCs' height is too small. Multiple options were considered to ensure that the images can be used with ResNet models, such as:

1. Use 224 mel-spaced bins and 224 cepstral coefficients.
2. Before the images are passed to the model, apply zero padding to the height dimension.
3. Change the parameters of the inner layers of the ResNet model.
4. Change the parameters of the first layer of the ResNet model.

The number of mel-spaced bins and the number of cepstral coefficients that are used to calculate the mel-spectrograms and the MFCCs determine the amount of detail in the frequency dimension. For speech-related tasks, 20-40 mel-spaced bins and 13-39 cepstral coefficients are usually used. Increasing the number of bins and coefficients to 224, increases the computational complexity and forces the parameters to be higher than the norm. It also prevents hyper-parameter tuning for the audio features which could prevent the model from finding the optimal solution.

Padding the images with 0s to match the desired shape is generally a good option. It introduces little computational complexity and ensures that one can perform hyper-parameter tuning on the audio features. The main concern with padding the images is that most of the image ends up being padded with 0s and this could influence features that the convolutional layers can extract to learn the wrong patterns in the data.

The generic ResNet has predetermined parameters for each layer including the output channels, kernel size, and the amount of padding and striding. These parameters have been fine-tuned to deliver the best features and are used to ensure that results are reproducible. By changing the parameters of the inner layers, the ResNet becomes less consistent with the rest of the ML space. This approach was attempted, but it was found that by changing the parameters, the computational complexity doubled.

The approach that was chosen for this project is changing only the first layer of the ResNet model. This was done to allow audio feature hyper-parameter tuning, allow for

optimal feature selection during convolution, and minimise the difference between this ResNet and the generic ResNet, whilst introducing minimal computational complexity.

The first layer of the ResNet model is a convolutional layer with $output\_channels = 64, stride = 2, padding = 3$ and $kernel = (7 \times 7)$. When the input data shape is $(224 \times 224)$ the resulting output dimension is $(64 \times 112 \times 112)$. When the input data shape is smaller $(224 \times n)$, the convolutional layer's parameters are changed to $output\_channels = 64, stride = (2, 1), padding = (3, padding\_height)$ and $kernel = (7 \times 1)$, where $padding\_height = (desired\_height - input\_size)//2$ and the desired height corresponds to the output of the generic ResNet's first layer. The $input\_size$ corresponds to the height of the input images $\therefore input\_size = num\_mel\_bins$ or $input\_size = num\_cepstrums$. As shown in Figure 5.6, the rest of the ResNet's parameters remain the same.

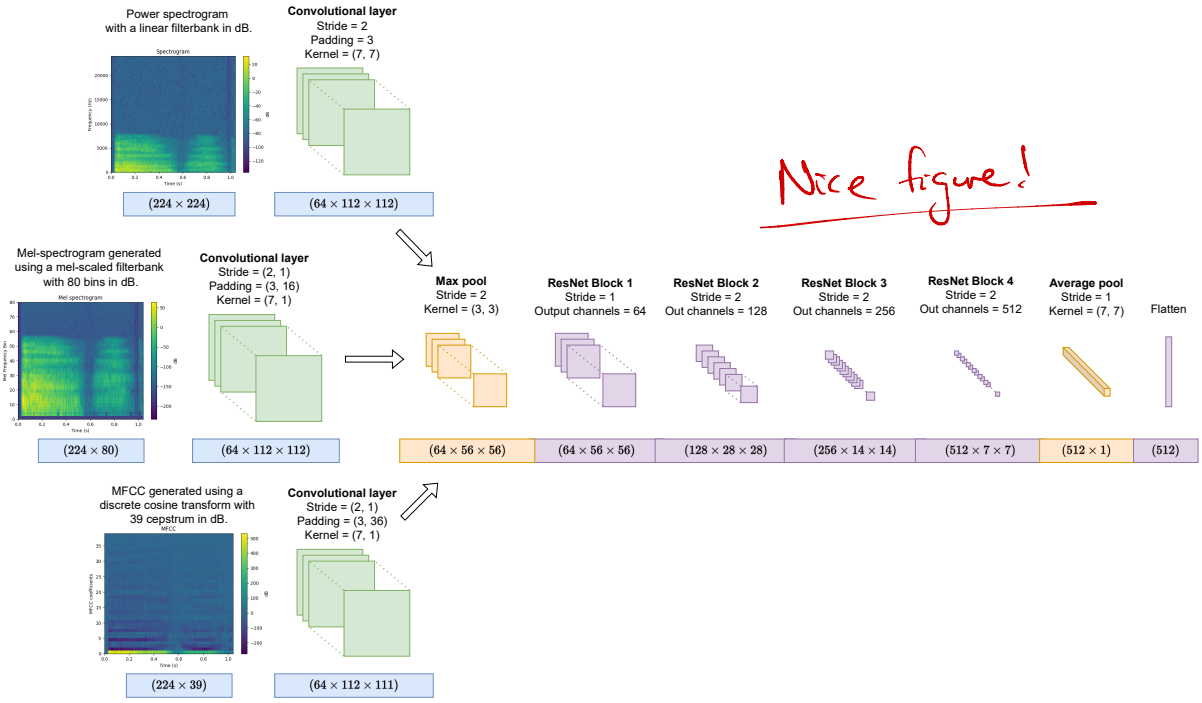


**Figure 5.6:** ResNet with altered first layer based on input data shape. Spectrogram images with shape $(224 \times 224)$ are already in the right shape, thus the first layer is the same as a generic ResNet. Mel-spectrograms and MFCCs have smaller heights so the first layer has a $(7 \times 1)$ kernel, stride in only the x-axis and padding to match the output dimension of a generic ResNet's first layer. The rest of the layers are the same as the generic ResNet layers.

The results are computed for logistic regression using spectrograms, mel-spectrograms and MFCCs. Outer fold 0 is used as the test set (not used currently), and the rest is used as cross-validation. The search space includes $learning\_rates = [5e-5, 5e-6], batch\_sizes = [64, 128], numbers\_epochs = [5, 10]$.

| Dev set | Learning rate | Batch size | Number of Epochs | AUC |
|---|---|---|---|---|
| 1 | 5e-6 | 64 | 10 | 0.616909 |
| 2 | 5e-6 | 128 | 5 | 0.569016 |
| 3 | 5e-6 | 64 | 10 | 0.579347 |
| 4 | 5e-6 | 64 | 5 | 0.583943 |
| 5 | 5e-6 | 64 | 10 | 0.616451 |
| 6 | 5e-5 | 64 | 10 | 0.617115 |
| 7 | 5e-5 | 64 | 10 | 0.602839 |
| 8 | 5e-6 | 64 | 10 | 0.559688 |
| 9 | 5e-6 | 64 | 5 | 0.570128 |

**Table 5.1:** Logistic regression hyper-parameter optimisation using spectrograms for the Hyfe dataset. Mean=0.591 and standard deviation=0.022.

| Dev set | Learning rate | Batch size | Number of Epochs | AUC |
|---|---|---|---|---|
| 1 | 5e-5 | 128 | 10 | 0.632878 |
| 2 | 5e-5 | 128 | 10 | 0.577857 |
| 3 | 5e-5 | 128 | 10 | 0.560854 |
| 4 | 5e-5 | 128 | 5 | 0.566853 |
| 5 | 5e-5 | 64 | 10 | 0.550967 |
| 6 | 5e-6 | 128 | 5 | 0.575317 |
| 7 | 5e-5 | 64 | 5 | 0.557486 |
| 8 | 5e-6 | 128 | 10 | 0.547169 |
| 9 | 5e-5 | 128 | 5 | 0.603378 |

**Table 5.2:** Logistic regression hyper-parameter optimisation using mel-spectrograms for the Hyfe dataset. Mean=0.575 and standard deviation=0.026.

| Dev set | Learning rate | Batch size | Number of Epochs | AUC |
|---|---|---|---|---|
| 1 | 5e-5 | 64 | 5 | 0.620538 |
| 2 | 5e-5 | 128 | 5 | 0.601781 |
| 3 | 5e-5 | 64 | 10 | 0.57132 |
| 4 | 5e-6 | 64 | 5 | 0.572148 |
| 5 | 5e-6 | 64 | 10 | 0.558537 |
| 6 | 5e-5 | 64 | 10 | 0.596891 |
| 7 | 5e-5 | 64 | 10 | 0.59752 |
| 8 | 5e-5 | 64 | 10 | 0.565041 |
| 9 | 5e-5 | 64 | 10 | 0.593484 |

**Table 5.3:** Logistic regression hyper-parameter optimisation using MFCCs for the Hyfe dataset. Mean=0.586 and standard deviation=0.019.