Deep Learning Based Methods for Tuberculosis Cough Classification

Geoffrey Frost

Thesis presented in partial fulfilment of the requirements for the degree of
Master of Engineering (Electronic) in the Faculty of Engineering at
Stellenbosch University.

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

March 2023

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2023

Abstract

Automated systems for disease identification have the potential to streamline the patient diagnosis process and provide insight to physicians. Exploratory studies have developed such systems for tuberculosis (TB) which rely on the cough audio signal produced by patients to determine, with the use of statistical classifiers, if they may have TB or not. Although these studies were small and the algorithms developed rudimentary, promising results were achieved. In this study, we build upon existing work and investigate the application of various deep learning -based approaches for TB cough classification. Since such systems would eventually be deployed in a multitude of environments and to increase the size of the dataset which is useful for model development, we combine the datasets used in previous studies. Multiple classifiers are developed, which include architectures based on bidirectional long short-term memory networks (BiLSTM), convolutional neural networks (CNN), attention and transformers. Additionally, the application of various large pre-trained models (ResNet, Audio Spectrogram Transformer and wav2vec2.0) to TB cough classification is investigated. Moreover, we develop a unique cough classification pre-training task to better initialise model parameters. Substantial classification performance improvements are observed compared to the previous best methods. In particular, the pre-trained BiLSTM architecture achieved relative improvements in AUC and EER of 9.33% and 64.67% respectively compared to the baseline system. More generally, the use of pre-training almost always improved the performance of these metrics, and always lead to better generalisation, observed by a reduction in metric standard deviation across evaluation sets. Due to the cross-validation procedure used during development, the choice of decision thresholds was sub-optimal, which subsequently lead to poor sensitivity and specificity. This was typically worse with pre-training. However, even when using oracle decision thresholds, classifiers were unable to reach the WHO standards for such a diagnostic tool. Additionally, we conduct a brief investigation into patient identity as a confounding factor during training and subsequent deep learning-based mechanisms to inhibit its learning. Whilst we present clear evidence that models learn the identity of patients in conjunction with the underlying TB signal, its removal does not significantly impact performance. Lastly, using insights from feature importance experiments, attention weights analysis, and adversarial synthesis we provide clues regarding the origin and characteristics of the learnt TB signal in cough. Specifically, we use these methods to identify the most important frequency bands for classification, the importance of certain temporal regions in cough, and the distinct spectral characteristics between idealised TB and non-TB coughs.

Uittreksel

Geautomatiseerde stelsels vir siekte-identifikasie het die potensiaal om die pasiëntdiagnose-proses meer vaartbelyn te maak en insig aan dokters te verskaf. Verkennende studies het sulke stelsels vir tuberkulose (TB) ontwikkel wat staatmaak op die hoes-klank wat deur pasiënte geproduseer word om, met die gebruik van statistiese klassifiseerders, te bepaal of hulle moontlik TB het of nie. Alhoewel hierdie studies klein was en die algoritmes rudimentêr ontwikkel het, is belowende resultate behaal. In hierdie studie bou ons op bestaande werk en ondersoek ons die toepassing van verskeie diepleer-gebaseerde benaderings vir TB-hoesklassifikasie. Aangesien sulke stelsels uiteindelik in verskeie omgewings ontplooi sou word en om die grootte van die datastel wat nodig is vir modelontwikkeling te vergroot, kombineer ons die datastelle wat in vorige studies gebruik is. Veelvuldige klassifiseerders word ontwikkel, wat argitekture insluit wat baseer is op "bidirectional long short-term memory networks" (BiLSTM), "convolutional neural networks" (CNN), "attention" en "transformers". Daarbenewens word die toepassing van verskeie groot vooraf-opgeleide modelle (ResNet, Audio Spectrogram Transformer en wav2vec2.0) op TB-hoesklassifikasie ondersoek. Boonop dit, ontwikkel ons 'n unieke hoesklassifikasie-voorafopleidingstaak om modelkomponente beter te inisialiseer. Aansienlike klassifikasie prestasieverbeterings word waargeneem in vergelyking met die vorige beste metodes. In die besonder het die vooraf-opgeleide BiLSTM-argitektuur verbetering behaal in AUC en EER van 9.33% en 64.67% onderskeidelik in vergelyking met die basislynstelsel. Oor die algemeen het die gebruik van vooropleiding byna altyd die prestasie van hierdie maatstawwe verbeter, en het altyd gelei tot beter veralgemening. Dit is waargeneem deur 'n vermindering in metrieke standaardafwyking oor die evalueringsstelle. As gevolg van die kruisvalideringsprosedure wat tydens ontwikkeling gebruik is, was die keuse van besluitdrempels sub-optimaal, wat vervolgens gelei het tot swak sensitiwiteit en spesifiekheid. Dit was gewoonlik slegter met vooropleiding. Selfs wanneer orakelbesluitdrempels gebruik word, kon klassifiseerders egter nie die WGO-standaarde vir so 'n diagnostiese hulpmiddel bereik nie. Daarbenewens doen ons 'n kort ondersoek na pasiëntidentiteit as 'n verwarrende faktor tydens opleiding en daaropvolgende diepleer-gebaseerde meganismes om die leer daarvan te verhoed. Alhoewel ons duidelike bewyse aanbied dat modelle die identiteit van pasiënte leer in samewerking met die onderliggende TB-klanksein, het die verwydering daarvan nie 'n noemenswaardige impak op prestasie nie. Laastens, deur gebruik te maak van insigte van kenmerkbelang-eksperimente, aandaggewigte-analise en teenstrydige sintese, verskaf ons leidrade aangaande die oorsprong en kenmerke van die aangeleerde TB-klanksein in ‘n hoes. Spesifiek, ons gebruik hierdie metodes om die belangrikste frekwensiebande vir klassifikasie te identifiseer, asook die belangrikheid van sekere temporale streke in ‘n hoes, en die duidelike spektrale kenmerke tussen ‘n geïdealiseerde TB en ‘n nie-TB hoes.

Acknowledgements

I would like to express my thanks to the following people and organisations, without which this work would not have been possible. Firstly, my supervisor Prof. Nielser for his tireless guidance, feedback, support, and for affording me this opportunity. The members of the DSP lab, particularly the cough & speech group, who were always willing to bounce ideas and provide technical insight. Lastly, my friends and family, who were a source of unwavering support and encouragement. I gratefully acknowledge funding from the EDCTP2 programme supported by the European Union (grant RIA2020I-3305, CAGE-TB).

Contents

Declaration i

Abstract ii

Uittreksel iii

List of Figures xi

List of Tables xvi

Nomenclature xix

1. Introduction 1

1.1. Research Motivation . . . 2
1.2. Project scope and contributions . . . 2
1.3. Published outputs . . . 3
1.4. Thesis overview . . . 3

2. Literature review 5

2.1. Tuberculosis . . . 5
2.1.1. Pathology and transmission . . . 6
2.1.2. Symptoms and diagnosis . . . 6
2.2. Cough classification . . . 6
2.2.1. Acoustic probing . . . 7
2.2.2. Analysis of cough . . . 7
2.2.3. Wet/Dry . . . 8
2.2.4. Pneumonia . . . 9
2.2.5. COVID-19 . . . 10
2.2.6. Summary . . . 13
2.3. TB cough classification . . . 13
2.3.1. Datasets . . . 14
2.3.2. Experimental setup . . . 14
2.3.3. Results . . . 16
2.4. General acoustic analysis . . . 18
2.4.1. Acoustic scene classification . . . 18

2.4.2. Keyword spotting . . . 19
2.4.3. Automatic speech recognition . . . 20
2.5. Summary . . . 21

3. Background 22
3.1. Feed-forward Neural Network . . . 22
3.2. Convolutional Neural Network . . . 24
3.2.1. Convolutional layers . . . 24
3.2.2. Stride and padding . . . 26
3.2.3. Convolutional block . . . 26
3.2.4. Residual connections . . . 27
3.3. Long Short-Term Memory Networks . . . 28
3.3.1. Recurrent Neural Networks . . . 28
3.3.2. LSTM network . . . 29
3.4. Attention . . . 31
3.5. Transformer encoder . . . 34
3.6. Activation functions . . . 34
3.7. Training . . . 35
3.7.1. Loss functions . . . 35
3.7.2. Back-propagation . . . 37
3.7.3. Optimisation . . . 37
3.7.4. Over-fitting . . . 39
3.8. Acoustic features . . . 42
3.8.1. Linear filterbank energies . . . 42
3.8.2. Mel-spectrograms . . . 42
3.8.3. Mel frequency cepstral coefficients . . . 43
3.8.4. Feature normalisation . . . 44
3.9. Summary . . . 45

4. Datasets 46
4.1. Brooklyn dataset . . . 46
4.2. Wallacedene dataset . . . 46
4.2.1. Data cleaning . . . 47
4.3. Combined dataset . . . 48
4.4. “Cough or Not” dataset . . . 49
4.5. Summary . . . 50

5. Pre-trained Networks 51
5.1. ResNet . . . 51
5.1.1. Motivation . . . 52

5.2. Audio Spectrogram Transformer . . . 53
5.2.1. Motivation . . . 54
5.3. wav2vec2.0 . . . 55
5.3.1. Motivation . . . 56
5.4. Summary . . . 57

6. Experimental setup 58
6.1. Architectures . . . 58
6.1.1. Linear Regression . . . 59
6.1.2. BiLSTM . . . 60
6.1.3. BiLSTM-Attention . . . 61
6.1.4. CNN . . . 62
6.1.5. Transformer . . . 64
6.1.6. ResNet . . . 65
6.1.7. Audio Spectrogram Transformer . . . 66
6.1.8. wav2vec2.0 . . . 67
6.2. Pre-training . . . 68
6.3. Acoustic features . . . 69
6.3.1. Data augmentation . . . 70
6.3.2. Sequential forward search . . . 72
6.4. Datasets, evaluation and training . . . 73
6.4.1. Dataset splits . . . 73
6.4.2. Performance Metrics . . . 75
6.4.3. Training . . . 78
6.4.4. Model selection and evaluation . . . 78
6.5. Summary . . . 79

7. Experimental Results 80
7.1. Optimal meta-parameters . . . 80
7.2. Classifier performance . . . 81
7.2.1. BiLSTM . . . 81
7.2.2. BiLSTM-Attention . . . 82
7.2.3. CNN . . . 82
7.2.4. Transformer . . . 83
7.2.5. ResNet . . . 84
7.2.6. AST . . . 84
7.2.7. wav2vec2.0 . . . 84
7.3. Discussion . . . 85
7.3.1. Model size . . . 87
7.3.2. Decision thresholds . . . 87

7.3.3. Oracle decision thresholds . . . 88
7.3.4. Sequential forward search . . . 89
7.4. Comparison to previous studies . . . 91
7.5. Summary . . . 92

8. Identity Confounding 94
8.1. Comparison to the null hypothesis . . . 94
8.1.1. Experimental setup . . . 95
8.1.2. Results and discussion . . . 96
8.2. Exploring identity-learning mitigation techniques . . . 97
8.2.1. Experimental setup . . . 98
8.2.2. Domain adversarial training . . . 99
8.2.3. Triplet loss . . . 101
8.2.4. GE2E loss . . . 102
8.2.5. Results and discussion . . . 103
8.3. Summary . . . 104

9. Acoustic Signature of Cough 105
9.1. SFS feature importance . . . 105
9.2. Temporal analysis . . . 105
9.3. Idealised coughs . . . 107
9.3.1. Adversarial synthesis . . . 107
9.3.2. Experimental setup . . . 108
9.3.3. Results and discussion . . . 109
9.4. Summary . . . 111

10. Conclusion 112
10.1. Summary and conclusions . . . 112
10.2. Future work . . . 114

Bibliography 115

A. Published work 124
A.1. Interspeech 2022 . . . 124

B. Dataset splits 130
B.1. Fold 1 . . . 130
B.1.1. Test set . . . 130
B.1.2. Inner fold 1 development set . . . 130
B.1.3. Inner fold 2 development set . . . 130
B.1.4. Inner fold 3 development set . . . 130

B.1.5. Inner fold 4 development set 130
B.2. Fold 2 131
B.2.1. Test set 131
B.2.2. Inner fold 1 development set 131
B.2.3. Inner fold 2 development set 131
B.2.4. Inner fold 3 development set 131
B.2.5. Inner fold 4 development set 131
B.3. Fold 3 131
B.3.1. Test set 131
B.3.2. Inner fold 1 development set 131
B.3.3. Inner fold 2 development set 132
B.3.4. Inner fold 3 development set 132
B.3.5. Inner fold 4 development set 132

C. The limitations of nested cross-validation 133
C.1. The problem 133
C.2. Discussion 135
C.2.1. Examples 135

D. Wallacedene noise categorisation 137

List of Figures

2.1. An illustration a cough and the corresponding region of the respiratory system from which it originates from. Observe how the initial sound originates from the lung airways (bronchi) whilst the second is as a result of reverberation in the voice box (Larynx). Reproduced from [1]. 8
2.2. The nested k-fold cross-validation strategy used to develop and test classifiers in [2] and [3]. Reproduced from [3]. 17

3.1. A simple example of a three layered feed-forward network consisting of an input layer (yellow), a hidden layer (orange) and a output layer (red). Geen highlights the input feature vector which is fed into the first layer. 23
3.2. The output of the 2-D convolution between a 3×4 input and a 2×2 kernel. Each time the kernel is stepped across the input the element-wise product between the kernel and corresponding overlapped region of the input is computed and summed. 25
3.3. A 3×3 edge detection kernel is applied to an image of a capybara. 25
3.4. Typical structure of a convolutional block, consisting of a convolutional layer, non-linear activation function, and an optional pooling layer. 27
3.5. Depiction of a residual block and its skip-connection. The input to the block is summed with the output. 27
3.6. A single RNN cell is shown, as well as an unfolded RNN in time showing the parsing of state information from previous time steps. 28
3.7. A single LSTM cell, showing in detail the computations performed with each gate's (where red, blue, and green highlight the forget, input, and output gates respectively) output and the cell state. In addition, a high level representation of an LSTM unfolded in time. 30
3.8. BiLSTM unfolded in time. 31
3.9. Attention cell, where $\boldsymbol{X}_q$ $\boldsymbol{X}_k$ $\boldsymbol{X}_v$ are the set of vectors to be used as queries, keys and values. Note in self-attention $\boldsymbol{X}_q = \boldsymbol{X}_k = \boldsymbol{X}_v$. 32
3.10. Visualisation of a practical attention application to cough. 33
3.11. A single transformer encoding block, whereby $\boldsymbol{X}$ has already been passed through a positional encoding. Dashed lines indicate residual skip connections. 35
3.12. Commonly used activation functions. 36

3.13. Plot of the cross-entropy loss compared to the predicted probability for a given class. As the confidence increases, the loss decays exponentially. . . . 37
3.14. Single parameter model updates (red arrows) for various learning rates and subsequent observed effects during training. Note we omit the loss functions dependence on $\boldsymbol{x}$ for simplicity. 39
3.15. Stylised learning rate sweep, illustrating how the optimal learning rate is selected. Note after each step model parameters are updated and the learning rate is increased. 40
3.16. An example of a linearly scaled filter bank. Each colour represents an individual filter. 43
3.17. Depiction of the mel scale. 43
3.18. Mel scaled filter banks. Each colour represents an individual filter. 44

5.1. An illustration of the 34-layer ResNet architecture (ResNet-34). Dotted shortcuts indicate an increase in channel dimension. Reproduced from [4]. . 52
5.2. A residual building block used in ResNet-18/34 (left) and the bottleneck residual building block used in ResNet-50/101/152 (right). Reproduced from [4]. 52
5.3. High-level diagram of AST. A 2D spectrogram is split up and linearly projected into patch embeddings where an additional classification token is prepended. The resulting sequence is combined with learnable positional embeddings and fed into a transformer encoder. The output corresponding to the classification token is then fed into a linear layer for classification. Reproduced from [5]. 54
5.4. Visualisation of reducing the dimension of a matrix along one axis, and interpolating it along another. Red squares indicate interpolated samples where $\tilde{\boldsymbol{e}}_{i,j}$ is the mean between $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$. 54
5.5. Illustration of the wav2wec2.0 architecture. A raw waveform is fed into a convolutional speech representation extractor, the output of which is used by a transformer-based context network. 56

6.1. Network diagram for the proposed BiLSTM architecture. The input feature vector is fed into a set of stacked BiLSTMs. The last embeddings in both directions are concatenated followed by a fully connected layer for classification. 60

6.2. Illustration of an input sequence $\boldsymbol{X}$ of length n with m padded samples being fed into a single BiLSTM layer and the formation of the single vector $\mathbf{q}$. The last output of the forward LSTM that does not correspond to a padded input $\boldsymbol{H}_f^{(n-m)}$ is concatenated with the last output of the backward cell $\boldsymbol{H}_b^{(n)}$, which because it processes the padded elements fist, may include information extracted over the padded region. 61
6.3. Network diagram for the proposed BiLSTM-Attention architecture. The input feature vector is fed into a set of stacked BiLSTMs. The last forward and backward representations in both directions are concatenated to form a query vector $\mathbf{q}$ which is parsed to an attention layer together with keys and values from the concatenated sequence of forward and backward representations ($\boldsymbol{H}_\mathrm{f}$ and $\boldsymbol{H}_\mathrm{b}$ respectively) outputted by the BiLSTM encoder. The attention mechanism outputs a single embedding which is the weighted sum of $\boldsymbol{H}$ by attention score relative to $\mathbf{q}$. This is then parsed to a fully connected network for classification. 62
6.4. Network diagram for the proposed CNN-based architecture. Each convolutional block is highlighted with a solid colour, where pink denotes the input block, and orange the convolutional blocks which will be stacked. Note k, s and c above each block denote the kernel size, stride, and the number of filters/channels respectively. 63
6.5. High-level diagram for the proposed transformer architecture. $\boldsymbol{X}$ is upsampled in the feature axis, with a single linear layer (same layer used for each element in the sequence) before being positionally encoded. This in then fed through β_n transformer encoding modules, after which the CLS token is used for classification. 65
6.6. wav2vec2.0 decoder structure used in [6]. Context representations are fed through two 1D convolutional blocks, and then a fully connected layer for classification. 68
6.7. Illustration of time warping being applied to a spectrogram (top) and the resulting transformation (bottom). The dashed red line indicates the selected anchor point [7]. 71
6.8. Example of frequency (top) and time (bottom) masking [7]. 71
6.9. Visualisation of nested k-fold cross-validation used in this work. The dataset is divided into three non-overlapping subsets (or three outer folds). Each training set is further divided in four non-overlapping subsets (or inner folds). These are used to optimise architecture parameters for that specific outer fold. This is repeated for each inner and each outer fold. 74

6.10. A confusion matrix depicting the ground truth and predicted labels for TP, TN, FP, and FN decisions. Cells highlighted in green indicate metrics where higher values indicate a better classifier, whereas red indicates metrics where lower values infer better performance. 75
6.11. ROC curves for two systems: A and B. 77
6.12. Representation of FPR and FNR as the decision threshold is increased. The error rate where FPR and FNR cross is labelled the EER. 77

7.1. Mean ROC curves across the outer folds for three selected architectures (wav2vec2.0, Transformer, and BiLSTM) and the LR baseline for both base (left) and pre-trained variants (right). 87
7.2. Mean development AUCs for each outer fold as the optimal subset of acoustic features is increased during SFS. 90
7.3. Mean BiLSTM test and development AUCs for each fold as the optimal subset of acoustic features is increased during SFS. 92

8.1. Patient-wise colour-coded t-SNE dimensionality reduction of d-vector speaker embeddings extracted using patient coughs. 95
8.2. Distributions of AUC scores given the experimental conditions described. . 98
8.3. A typical domain adversarial neural network. Note the feature extractor (green), the primary classifier (blue) and domain classifier (pink) which is connected to the feature extractor via a gradient reversal layer. 100
8.4. Visual representation of the relationship between the anchor $\mathbf{e}$, positive $\mathbf{e}^+$, and negative $\mathbf{e}^-$ pairs for a given triplet. Negative samples are pushed away from the anchor passed the margin, whilst positive samples are brought closer. 101

9.1. Acoustic feature center frequencies for the first ten optimal features discovered by SFS for each fold. 106
9.2. Various cough mel-spectrograms and their respective attention weights are shown above them. The first two rows correspond to coughs from the same TB patient, whilst the last two correspond to coughs from a $\overline{\text{TB}}$ patient. Reproduced from [8]. 107
9.3. Illustration of the loss back-propagation during adversarial synthesis training. 108
9.4. Idealised mel-spectrograms and mean spectral power for the synthesised coughs produced by the BiLSTM-Attention architecture. Reproduced from [8]. 109
9.5. Idealised waveforms for a TB negative (top) and TB positive (bottom) cough produced by the wav2vec2.0 architecture. 110
9.6. Mel-spectrograms of and mean spectral power of synthesised cough waveforms produced by the wav2vec2.0 architecture. 110

C.1. Two partitions of a nested k-fold cross validation strategy $\mathbf{S}_a$ and $\mathbf{S}_b$ are visualised. Blue samples indicate training data points that overlap between a nested fold and another outer fold, while red indicates dev/test samples that overlap between the same nested fold and outer fold. 134

List of Tables

2.1. Summary of performance metrics (sensitivity and specificity) for proposed models in various cough classification literature described in this section. Table entries marked with a '*' denote experimental setups that have not distinguished between a test and a development set. 13
2.2. Test set classification performance as presented in [2]. 17
2.3. Test set classification performance as presented in [3]. 18

4.1. Brooklyn dataset description. 47
4.2. Wallacedene dataset description. 47
4.3. Combined dataset description. 48
4.4. "Cough or Not" dataset description for each audio source. 49

6.1. Dimensionalities considered for various acoustic feature vectors. 69
6.2. Train and test set splits of the combined dataset for the three outer folds (fold 1 to 3 from left to right). 73
6.3. Batch sizes used when training various architectures. 78

7.1. Mean development AUC for the first outer-fold for the two investigated augmentation techniques. Namely, speed-perturbation (S-P) and SpecAugment. 80
7.2. Mean and standard deviation of AUC and the decision thresholds ($\bar{\gamma}$) during 4-fold cross-validation. 81
7.3. BiLSTM test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively. 82
7.4. BiLSTM-Attention test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively. 82
7.5. CNN test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively. 83

7.6. Transformers test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with “base” and “pt” respectively. . . . 83
7.7. ResNet test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with “base” and “pt” respectively. . . . 84
7.8. AST test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with “base” and “pt” respectively. . . . 84
7.9. wav2vec2.0 test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with “base” and “pt” respectively. . . . 85
7.10. Summary of test set performance across all architectures, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. . . . 86
7.11. Investigated architectures parameter count and respective mean test set AUC with pre-training with exception of the LR baseline. . . . 88
7.12. Mean and standard deviation of EER decision thresholds γ across outer folds for selected architectures. . . . 88
7.13. Oracle specificity and sensitivity of selected classifiers when tuning decision thresholds to meet the WHO’s specificity and sensitivity requirements (0.7 and 0.9 respectively). . . . 89
7.14. Summary of test set performance after performing SFS, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. . . . 91
7.15. Comparison of test set performance after performing SFS on the same dataset splits (fold 1) for the BiLSTM architecture for two different experimental code-bases. . . . 91

8.1. Monte-Caralo simulations and their respective experimental constraints. Simulation A and C represented the null hypothesis when patient overlap is and is not allowed respectively. . . . 96
8.2. Mean and standard deviation of AUC for estimated probability densities generated given conditions A, B, C, and D respectively. . . . 97

8.3. Mean and standard deviation of AUC for the baseline architecture and the inclusion of patient-learning mitigation strategy. 103

D.1. Wallacedene cough recording notes per patient, categorised into three disturbance categories: "None", "Some" and "Significant". 137

Nomenclature

TB	tuberculosis
MFCC	mel-frequency cepstral coefficient
LSTM	long short-term memory network
CNN	convolutional neural network
FNN	feed-forward neural network
WHO	World Health Organisation
PCR	polymerase chain reaction
LR	logistic regression
LOO	leave-one-out
CV	cross-validation
SMOTE	synthetic minority over-sampling technique
KNN	k-nearest neighbour
SVM	support vector machines
ROC	receiver-operator characteristic
AUC	area under the ROC curve
LFB	linear filter bank
STFT	short-time Fourier transform
SFS	sequential forward search
KWS	keyword spotting
ASR	automatic speech recognition
AST	audio spectrogram transformer
ViT	vision transformer
BiLSTM	bidirectional long short-term memory units
RNN	recurrent neural network
tanh	hyperbolic tan
ReLU	rectified linear unit
SGD	stochastic gradient decent

DCT	discrete cosine transform
CTC	connectionist temporal classification
EER	equal error rate
TP	true positive
TN	true negative
FP	false positive
TN	true negative

Chapter 1

Introduction

Tuberculosis (TB) is a highly infectious disease. In 2021, 10 million people were reported to have developed tuberculosis (TB), of whom 1.5 million died. As a result, TB was the second most lethal infectious disease globally ranking above HIV/AIDS and just below COVID-19 [9]. The majority of TB cases occur in developing nations where access to public health care is limited by complex socio-economic factors, making it difficult to identify and control the spread of the disease, and resulting in patients receiving improper care [10].

Current TB diagnosis tools are costly and typically require specialised skills and infrastructure to apply. The developing world's primary health care facilities often lack the required funds to employ such diagnostic techniques needed for rapid screening and have to rely on more time-consuming and rudimentary (but cost-effective) methods such as sputum analysis [9]. This results in long diagnosis lead times, allowing TB outbreaks to spread rapidly through communities. As such, there is a need for a low-cost, rapid TB screening method which can be used to identify patients who are likely to be TB positive and refer them for more robust confirmatory testing.

Cough is one of the most predominant symptoms of respiratory diseases. Due to the assortment of pathological and physiological conditions that can cause cough, it is useful to be able to distinguish between each ailment's cough characteristics. Since the invention of the stethoscope, the sound of the lungs, particularly when coughing, has been used as a diagnostic tool for respiratory diseases by physicians [11]. Whilst the analysis of cough is usually a manual process which relies on the physician's experience, automated analysis of cough provides a potential avenue for faster and more accurate diagnostics. Through both spectral and time domain analysis, it has been demonstrated that cough can provide insight into the characteristics of the lungs under different respiratory conditions [1]. Although published research covering automatic cough classification is currently limited, some studies have shown promising results in distinguishing between wet and dry coughs [12–14], classifying pneumonia [15, 16], and more recently detecting COVID-19 [17–20].

TB is predominately a respiratory disease, typically resulting in patients developing a chronic cough. Despite no humanly audible difference, it has been shown in the literature that it is possible to distinguish between the coughs of TB patients and healthy controls by utilising simple statistical classifiers [2]. More recently, these same methods have been evaluated on a dataset that aims to mimic *real-world* conditions, whereby coughers all suffer from some lung ailment that is in some cases TB [3].

1.1. Research Motivation

Previous TB cough classification studies largely focused on applying simple linear classifiers. These classifiers required fixed dimensional inputs, which were achieved by computing the frame-wise average of an acoustic feature vector and thus disregarded temporal information. Whilst these systems demonstrated that they can perform surprisingly well, a thorough investigation has not been conducted into the application of more sophisticated approaches which retain and utilise the temporal characteristics of cough. Deep learning -based systems, which achieve state-of-the-art performance in many other acoustic-based problems, can learn highly non-linear relations in the feature space, and in some architectures, are designed to specifically capture time-dependent information. As such, we aim to investigate the application of deep learning architectures to TB cough classification and compare their performance to the best methods used in the literature.

Moreover, the experimentation presented in the literature was conducted with datasets collected from a single recording domain, and as such do not test how classifiers generalise to multiple recording conditions. This is an important consideration when aiming to use such a classifier in a real-world scenario, where such a diagnostic tool will be used in a variety of circumstances. Consequently, we conduct experiments using a dataset comprising cough recordings collected from two domains (a combination of the datasets presented in the literature) to contribute toward a better understanding of classifier performance in such a scenario.

Very little is known concerning what distinguishes seemingly similar coughs originating from TB-positive and TB-negative patients. Analysis of the feature space that is learnt to be important by classifiers when distinguishing between such coughs may provide useful insight for researchers and aid in the design of systems in future work. As such we investigate techniques that aim to provide such insights.

1.2. Project scope and contributions

The scope of this project is limited to the application of the selected deep learning -based architectures to TB cough classification and comparing them to the previously used best methods. The investigated systems do not necessarily need to improve upon systems developed before this work. Whilst some work will be conducted to determine distinct characteristics of a TB cough, this is not a primary focus, and as such is not rigorously investigated.

This study contributes to an investigation into deep learning -based systems for TB cough classification. These include long short-term memory networks (LSTMs), convolutional neural networks (CNNs), attention, transformers and three large pre-trained networks. Some of these architectures substantially improve upon previously best methods. In addition, we investigate the application of a pre-training task to these investigated architectures and show it generally improves both model generalisation and performance for TB cough classification. We conduct a brief investigation into the impact of patient identity information present in cough on model generalisation during training. This includes the application of various techniques to inhibit the learning of patient identity through network adaptations and additional loss functions, concluding that its impact on performance is insubstantial. Further contributions include an adversarial technique used to synthesise idealised cough feature representations which are subsequently used to infer important characteristics of cough learnt to be important for classification. Additionally, a brief analysis of the important temporal regions in cough for classification is presented. Aspects of this work have been published in Interspeech 2022 [8], which can be found in Appendix A.

1.3. Published outputs

The following peer-reviewed conference paper has resulted from the work presented in this thesis:

- G. Frost, G. Theron, and T. Niesler,"TB or not TB? Acoustic cough analysis for tuberculosis classification," in *Proceedings of Interspeech*, 2022, pp. 2448–2452.

1.4. Thesis overview

This thesis is structured as follows. Chapter 2 provides a brief review of TB and the literature concerning cough analysis and classification, with emphasis on past studies completed on TB cough classification. An introduction to works of interest in other audio-processing domains is also provided. Chapter 3 introduces fundamental background in deep learning concepts and building blocks for the neural network -based architectures used in this work, namely feed-forward neural networks (FNNs), CNNs, recurrent networks (LSTMs), attention, and transformers. Additionally, an explanation of the acoustic feature representations chosen to be investigated is provided. Chapter 4 contains a detailed description of all data used in this work, which includes how data from previous studies was combined into the set used herein, and the composition of the dataset used for the auxiliary pre-training task. This is followed by detailed descriptions of pre-trained architectures considered in this study presented in Chapter 5. The details of the application of deep learning -based architectures to TB cough classification are presented in Chapter 6, which includes each investigated architecture, experiments, and experimental setup. The results of these experiments are presented in Chapter 7. An investigation into patient identity as a confounding factor in past TB cough classification systems is presented in Chapter 8, in addition to experiments which investigate the effect of inhibiting a classifier's ability to learn patient identity. Chapter 9 contains an analysis of the acoustic signature of cough which utilises, among other techniques, synthesised idealised cough representations with respect to trained systems. Lastly, Chapter 10 provides a summary of the work presented in this study, conclusions and an outlook on future work.

Chapter 2

Literature review

Cough is one of the most predominant symptoms of respiratory diseases. Due to the variety of pathological and physiological conditions that can cause cough, it is useful to be able to distinguish between each ailment's cough characteristics. However, many conditions result in acoustically similar coughs - making it challenging to correctly identify ailments without further testing. To this end, attempts have been made to leverage machine learning to automatically learn nuanced differences indiscernible to the human ear and classify coughs accordingly.

In this chapter, we first give a brief overview of the tuberculosis disease and describe typical symptoms and diagnostic tools. We then broadly discuss work completed in cough classification and describe the two previous studies that investigated TB cough classification in detail. Lastly, studies of interest and inspiration that are not directly linked to cough classification, but rather other acoustic tasks, are described. When discussing previous works (unless deemed necessary for immediate context) we do not describe exact architectures, feature extraction methods, or evaluation metrics in detail. However, aspects of the aforementioned that are of interest to this study are described in more detail in subsequent chapters. This includes relevant deep learning methods and acoustic feature vectors (Chapter 3) and evaluation metrics (Chapter 6).

2.1. Tuberculosis

TB is a highly infectious disease. In 2021 10 million people were reported to have TB, of whom 1.5 million died. These cases disproportionately occur in developing nations, with the majority of cases taking place in the densely populated regions of India, China, South East Asia and Africa [9]. In South Africa, it is estimated that almost 80% of the population is infected with latent TB, which refers to TB bacteria present in their bodies but asymptomatic with a chance of developing into active TB [21]. Further, in 2021 the World Health Organisation (WHO) estimated that there were at least $340,000$ active TB cases in South Africa (3.3% of the total active cases in the world) of which $55,000$ resulted in deaths, a mortality rate of 16% [9].

2.1.1. Pathology and transmission

Infection occurs when tubercle bacilli (the bacterium that causes TB) carrying droplets are inhaled and reach the lung's alveoli (nodules that branch off the bronchi responsible for re-oxygenating blood). The tubercle bacilli multiply and spread through lymphatic channels and the bloodstream, resulting in the development and facilitating the spread of TB bacteria throughout the body where the disease overcomes the immune system [22].

2.1.2. Symptoms and diagnosis

TB can affect a host of different parts of the body, resulting in the potential for multiple parallel infection sites. However, it is most common for TB to occur in the pulmonary system (lungs), resulting in patients developing a characteristic cough. The more rare extrapulmonary TB occurs when bacteria infect sites outside the lungs, which include the larynx, lymph nodes brain, kidneys, bones and joints. It is typical for extrapulmonary TB to accompany pulmonary TB in HIV patients [22].

Since TB can present itself with a variety of symptoms, there are several methods for diagnosing TB. The most common in South Africa is the analysis of sputum samples, skin tests, and chest X-rays. Sputum samples are viewed under a microscope, which even with a trained expert, has an accuracy $50 - 60\%$ [23]. Skin tests are used to test the sensitivity of a patient to the TB virus by injecting a small amount of tuberculin into the patient's arm. An analysis is conducted when the patient returns to the testing centre a few days later and requires a large degree of expertise to interpret the results, owing to the numerous factors that influence the outcome of the test (age, other illnesses present in the patient and immunological status) [24]. The more popular chest X-rays are used to detect the presence of TB manifestations in the lungs in an effective and fast manner. However, they require expert doctors to interpret results and expensive equipment, and can only detect pulmonary TB [25]. Lastly, an accurate, but substantially more expensive diagnostic tool are polymerase chain reaction (PCR) tests which identify DNA sequences associated with TB from a sputum sample. However, with each test costing R150 and an initial investment of R2.5 million required for the necessary lab equipment, it is not practical for large-scale screening [26].

2.2. Cough classification

Cough is a prevalent symptom of pulmonary disease, and is typically used in conjunction with other clinical information to diagnose the cause of illness. However, this usually subjective process leaves room for large degrees of error. Despite this, the use of cough in automated systems to aid in the precise diagnostics of pulmonary diseases is a largely under-researched field. Whilst [27] did not directly investigate cough, they demonstrated the possibility of airway pathology diagnosis through acoustic probing - indicating a difference in spectral response depending on the disease. Through both spectral and time domain analysis, [1] demonstrated that cough can provide insight into distinct characteristics of the lungs under different respiratory conditions. Whilst there is limited material on cough classification, a few studies have shown promising results. By combining both clinical data and coughing sounds, success has been achieved in distinguishing between wet/dry coughs - an important distinction in paediatric treatment [12–14]. Another body of work has focused on the classification of pneumonia in children suffering from a range of repository illnesses, concluding cough contained critical information on the lower respiratory tract relevant to accurate diagnosis [15, 16]. More recently, the abundance of COVID-19 related cough data being collected has enabled more sophisticated machine learning approaches to be applied to cough classification - with varying degrees of success [17–20]. We elaborate on each body of work in the sections that follow, and then present a summary of important overarching themes deduced from these works.

2.2.1. Acoustic probing

Whilst not strictly cough classification, [27] was able to determine easily transmittable frequencies that corresponded to certain airway pathology by introducing a controlled acoustic signal into the mouths of patients. In particular, by inputting a ramp of pure tone signals into patients' airways and measuring the response, the authors determined strong transmission at 650Hz for all subjects, whilst only healthy non-smokers had strong transmission at 2.5khz. Further detailed analysis indicated that lower frequency resonance is determined by characteristics of the trachea and bronchi whilst higher resonant frequencies are generated in the mucus-lined middle airway, which is typically where diseases manifest. The authors hypothesised that the dampening effect of increased mucus was the cause for decreased resonance at 2.5khz for subjects that were ill or smoked.

2.2.2. Analysis of cough

In [1], a thorough investigation was conducted into the characteristics of cough resulting from various lung ailments. Distinct differences in physiological and pathological coughs were observed. Power spectra from the former conformed to an exponential decay with an increase in frequency, whilst a linear decay was observed for the latter. Moreover, unique cough sound patterns from pathological conditions (bronchitis, laryngitis and tracheitis) were observed - indicating the potential for a distinction between acute chronic respiratory diseases based on the temporal nature of cough.

In addition to analysing the differences between cough sounds, the relationship between airflow changes measured from the mouth and the cough sound creation was also investigated. Typically, a cough is characterised by two high-energy bursts in quick

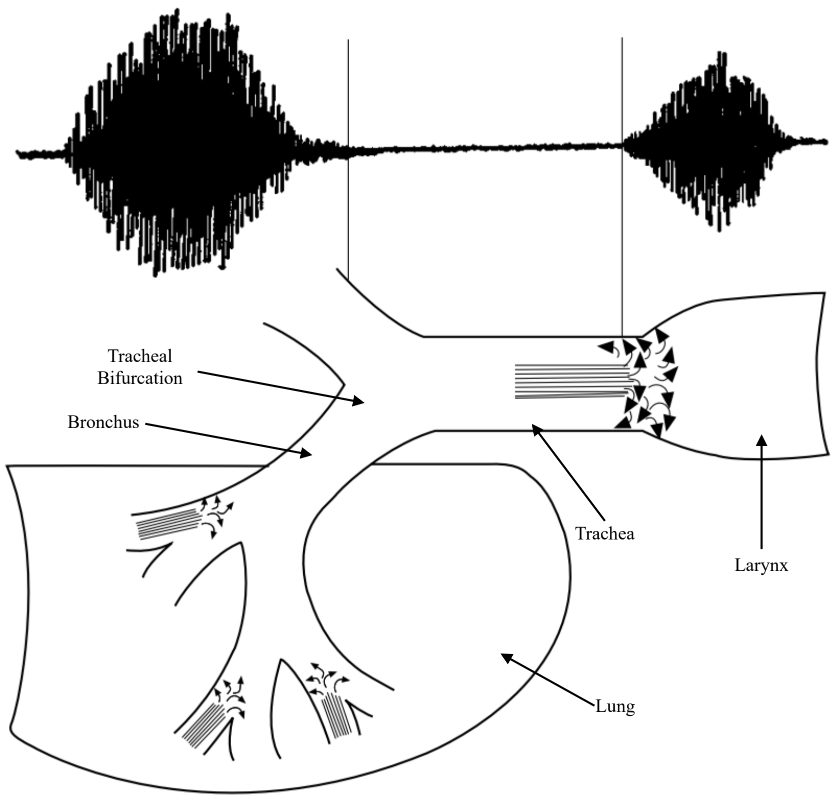


Figure 2.1: An illustration a cough and the corresponding region of the respiratory system from which it originates from. Observe how the initial sound originates from the lung airways (bronchi) whilst the second is as a result of reverberation in the voice box (Larynx). Reproduced from [1].

succession as illustrated in Figure 2.1. Motivated by peak values and differing vibrations in airflow curves, it was deduced that sound corresponding to the first peak in energy carries information regarding the pathological condition in airways peripheral to the level of the tracheal bifurcation whereas the second sound gives more information about the laryngeal area (around the larynx). The intermediate time between the two explosive sounds contains information regarding the trachea. For clarity, these areas are labelled in Figure 2.1.

2.2.3. Wet/Dry

Coughs are typically categorised into two groups: ‘Wet’ and ‘Dry’. This important distinction is useful in the diagnosis of respiratory diseases, for example, pneumonia (wet) and bronchiolitis (dry). The distinct difference between the two categories is a result of the presence (or absence) of mucus and the resulting change in acoustic signature. Typically, this distinction is made by a physician in an initial consultation period which afterwards the cough characteristic is not monitored. Approaches to automating this classification process aim to minimise subjectivity and enable long-term monitoring of a patient’s cough as treatment progresses. For instance, the progression of a wet cough to a dry cough could indicate an improved condition.

Whilst there is some work focused on feature extraction for manual classification [28,29], the authors of [14] were the first to attempt to automate wet/dry cough classification. They proposed a logistic regression (LR) classifier that utilised a feature vector consisting of several measurements: bispectrum score, non-gaussianity score, formant frequencies, log energy, zero-crossing rate, kurtosis and mel-frequency cepstral coefficients (MFCCs). They utilised a dataset consisting of 46 patients (178 individual cough episodes) diagnosed with diseases such as asthma, pneumonia, bronchitis and rhinopharyngitis. By performing greedy feature selection, an optimal subset of the original feature set was determined (23 features) and the resulting classifier achieved in a mean sensitivity and specificity of $79 \pm 9\%$ and $72.7 \pm 8\%$ respectively. Whilst the authors do describe a dataset partitioning strategy similar to k-fold cross-validation, they only ever mention train and test set splits. With the omission of any development or validation set, the validity of their results is brought into question. This body of work was furthered in [13] by utilising a larger dataset (76 patients), whereby explicit development (leave-on-out cross-validation (LOO CV)), and test strategies were described. Sensitivity and specificity of 84% and 76% respectively were reported.

2.2.4. Pneumonia

childhood pneumonia results in the death of 700,000 children around the world each year - the majority of which occur in underdeveloped nations [30]. In these regions, pneumonia outbreaks are often managed by community workers who lack the professional healthcare training required to adequately administer a timely diagnosis and provide the required treatment. Being a disease that resides in the lower respiratory tract, a key symptom of pneumonia is cough. As such, a small body of research exists to investigate the efficacy of automated pneumonia classification from recordings of a patient's cough.

The authors of [16] gathered a large dataset consisting of 91 paediatric patients, of which 63 had pneumonia and 28 had some other lung ailment (bronchiolitis, asthma, bronchitis, pharyngitis and laryngomalacia). The study was able to demonstrate that the automatic classification of pneumonia with cough sounds alone is feasible. A LR classifier was developed that utilises the exact feature set as described in Section 2.2.3 and the same optimal feature selection process. Whilst an investigation was also conducted into classification using both acoustic and clinical features, utilising cough alone a sensitivity and specified of 94% and 75% was achieved respectively.

This was furthered in [15], whereby a feed-forward neural network (FNN) was developed to perform the classification of pneumonia and asthmatic coughs, Once again using a similar feature vector as described by previous works. A small dataset consisting of 18 patients was used in combination with a LOO CV strategy to train and test the model. No distinction between a development and test set is made, resulting in a degree of uncertainty in their results. Nevertheless, sensitivity and specificity of 88.9% and 100.0% were reported.

2.2.5. COVID-19

Recently, with the SARS-CoV2 coronavirus (referred to as COVID-19) being declared a global pandemic by the WHO, large data collection efforts have been conducted to better understand its characteristics. Since COVID-19 is a disease that largely affects the respiratory system, these schemes have resulted in an abundance of cough recordings also being collected. This large increase in available data has enabled the application of more sophisticated data-driven machine learning methods for cough classification, namely, deep learning.

In [17] two datasets were used: the large Coswara dataset which collected cough recordings around the globe through an online portal (1171 participants) [31] and the much smaller Sarcos dataset collected in South Africa (44 participants). Due to its size, Sarcos was only used to simulate testing the developed classifiers on unseen recording domains and was not used for training or development. To address the large dataset imbalance between healthy and sick individuals (1027 vs 92 for Coswara and 26 vs 18 for Sarcos) the synthetic minority over-sampling technique (SMOTE) was applied. SMOTE synthesises new examples from the under-sampled class by randomly sampling an authentic sample and taking the random weighted average between one of its k nearest samples of the same class (typically determined by Euclidean distance) [32]. This is described in Equation 2.1, where $\boldsymbol{X}$ is the selected sample, $\boldsymbol{X}_k$ is one of the similar samples and u is uniformly distributed between 0 and 1.

$$\boldsymbol{X}_{\text{SMOTE}} = \boldsymbol{X} + u \cdot (\boldsymbol{X}_k - \boldsymbol{X}) \tag{2.1}$$

Several of architectures were experimented with, including various shallow linear models (LR, k-nearest neighbours (KNN) and support vector machines (SVM)) and deep neural networks, which included FNNs, convolutional neural networks (CNN), long short-term memory networks (LSTM) and ResNet-50 (a popular CNN-based architecture pre-trained on an image classification task [4]). The neural architectures all outperformed their shallow counterparts. Using a nested k-fold cross-validation technique, the authors optimised feature hyper-parameters (number of MFCC coefficients, Fourier transform length, temporal axis length) in addition to architecture parameters (batch size, learning rate, kernel size, model depth, etc.). Notably, through this strenuous process, the ResNet-50 architecture was able to achieve specificity and sensitivity of 98% and 93% respectively with an area under the receiver-operator characteristic curve (AUC-ROC, referred to as just AUC hereafter) of 0.976 when trained and evaluated on the Coswara dataset. In contrast, LR which is state-of-the-art in other domains achieved a sensitivity and specificity of 57% and 94% respectively with an AUC of 0.736. When tasked with applying these developed architectures trained on Coswara to an unseen recording domain, namely the Sarcos dataset, the LSTM architecture performed best achieving a specificity, sensitivity and AUC of 73%, 75%, and 0.7790 respectively. This was further improved by performing a sequential forward search (SFS, a feature optimisation process described in more detail in later sections [33]) to 96%, 91%, and 0.9380 respectively. Such impressive performance indicates that given sufficiently large quantities of data, deep architectures can improve upon the best methods used in the aforementioned other cough classification domains. Furthermore, the ability to transfer knowledge learnt from one recording domain to another, and still achieve strong performance indicates that given enough data models can generalise when tasked with cough classification.

The use of pre-training to improve the performance of these architectures was investigated in [34], whereby these same networks as those in [17] were trained on an auxiliary task before being fine-tuned for TB cough classification. A dataset consisting of 10 hours of various environmental noises (speech, sneezing, laughing, and throat cleaning) and coughing sounds was constructed and subsequently used in the pre-training step whereby models were trained to distinguish between coughs and the other environmental sounds. Marginal classification improvements were observed when deep models were evaluated on the in-domain Corswara dataset. However, for the much smaller Sarcos dataset, substantially improved performance was observed compared to non-pre-trained counterparts. Another important observation of the effects of pre-training was its impact on the reduction of the standard deviation across outer-fold test set metrics. This indicates that the pre-training of cough classifiers allows for better generalisation, which is an important factor should such classifiers be used as a screening tool.

Further demonstration of neural network's ability to learn effective latent representations for COVID-19 cough classification was shown in [19] with a dataset comprised of $3,621$ participants of which $2,001$ were confirmed to have the disease collected in India. The authors did not investigate the performance capabilities of different architectures but rather opted to optimise a single architecture based on ResNet-18 [4] and benchmark it against various shallow baselines (LR, SVM, and gradient-boosted trees). During training, two-second-long segments of audio were randomly sampled from a given patient's recording and a log mel-spectrogram was extracted (64 mel filter bins) which served as input to the CNN-based model. Numerous techniques were employed to increase model performance. First, background noise sampled from the ESC-50 dataset [35] (consisting of 2000 recordings of 50 classes) was superimposed on cough recordings. Next, a popular spectrogram-based augmentation technique is utilised - SpecAugment. SpecAugment performs frequency and time masking, and is been shown to improve model performance in a variety of acoustic tasks [7]. Additionally, the authors utilised a pre-training scheme. In a similar fashion to [34], the proposed architecture was trained on cough classification task combining the Freesound [36], FluSense [37] and Coswara datasets. Lastly, to prevent mislabelled coughs (due to the inaccuracies of COVID-19 testing) from hindering model performance, label smoothing (a logit regularisation technique) was applied. The classifier's performance was comparably modest compared to results presented in [17, 34], but the deep architecture still outperformed its shallow counterparts. The authors also acknowledged and presented the effects of label smoothing on model performance. In addition to resulting in an AUC increase from 0.65 to 0.68 (indicating label noise was hindering model performance), it drove the decision threshold substantially closer to 0.5 (0.02 to 0.422) reducing the model's class bias. The developed model was also applied to a case study which demonstrated that when using their classifier as a triage tool and selecting a threshold such that the model has high specificity (90%), they can increase testing capacity by 43% when assuming a disease prevalence of 5%.

Using an approach that deviates from previously discussed work, [20] proposed a method to aggregate both audio and clinical data to improve cough classification performance. Instead of extracting MFFCs, a continuous wavelet transform was performed. Wavelet transforms balance frequency resolution and time resolution by constructing an array of varied in length windowing functions to capture frequency information at specific bands (smaller windows for high frequencies, and larger windows for low frequencies). This is accomplished by replacing the typical windowing function with the complex conjugate of a wavelet function Φ, where s is a scaling factor, described in equation 2.2. Intuitively, smaller values for s result in a tighter wavelet, resolving higher frequency components with tighter time resolution, whilst larger values resolve lower frequency components with tighter frequency resolution.

$$F(\tau, s) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{\infty} f(t)\Phi^* \left(\frac{t-\tau}{s}\right) dt \tag{2.2}$$

The resulting scalogram served as the input to a CNN pre-trained as an autoencoder. During classification fine-tuning, the output of the encoder was concatenated with an encoding of available clinical data and fed into an FNN for classification. Experiments were conducted using COUGHVID [38], a large corpus of COVID-19 cough data. $1,116$ samples were used for training, whilst the test set consisted of 340 patients. The authors do not differentiate between development and test sets, which brings the robustness of their results into question. Nevertheless, specificity, sensitivity and precision of 0.81, 0.43 and 0.56 were reported respectively.

2.2.6. Summary

In the preceding sections, we presented studies that investigated the automated classification of coughing sounds for the purpose of disease classification. The specific classification tasks included wet/dry cough distinction, pneumonia detection, and COVID-19 detection. All three tasks show promising results, as highlighted by the performance summary of each presented in Table 2.1. Whilst results presented for wet/dry and pneumonia classification is promising, it is hard to make conclusions regarding the robustness of their results due to the small sizes of the datasets used. In contrast, COVID-19 studies used much larger datasets, increasing the robustness of their results. It is thus encouraging to see such impressive performance being reported which further serves as a strong indicator that cough can be used for respiratory disease classification. We also observe an overwhelming prevalence of MFCC features, despite these having been designed for speech processing. This potentially indicates that cough shares some similar characteristics to speech in relation to how information is carried in the frequency spectrum.

Table 2.1: Summary of performance metrics (sensitivity and specificity) for proposed models in various cough classification literature described in this section. Table entries marked with a '*' denote experimental setups that have not distinguished between a test and a development set.

Disease	Study	Model	Feature	Sens	Spec
Wet/Dry*	[14]	LR	MFCCs + metadata	0.79	0.73
Wet/Dry	[13]	LR	MFCCs + metadata	0.84	0.76
Pneumonia	[16]	LR	MFCCs + metadata	0.94	0.75
Pneumonia*	[15]	FNN	MFCCs + metadata	0.94	0.75
COVID-19	[17]	ResNet-50	MFCCs	0.93	0.98
COVID-19	[17]	LSTM (pre-trained)	MFCCs + SFS	0.91	0.96
COVID-19	[34]	ResNet-50 (pre-trained)	MFCCs	0.98	0.97
COVID-19	[34]	ResNet-50 (pre-trained)	MFCCs	0.96	0.72
COVID-19	[19]	ResNet-18 (pre-trained)	mel-spectrogram	0.90	0.31
COVID-19*	[20]	CNN (pre-trained)	scalogram + metadata	0.81	0.43

2.3. TB cough classification

We will now discuss the two studies that comprise the previous work completed on TB cough classification. As previously highlighted, there are several established clinical methods for detecting TB. However, these diagnostic tools either suffer from high cost, or from the need for highly specialised physicians or equipment. These limitations make them less accessible to high-risk populations and were the motivation for the first exploratory work into the viability of using coughs for automatic TB classification [2]. This was later extended, whereby the same model architectures were applied to a dataset collected in an environment more representative of the intended real-world deployment of such a system [3]. Since both of the aforementioned studies share nearly identical structures, we will detail the process followed by both in parallel.

2.3.1. Datasets

Whilst we give a more in-depth description of the datasets used in both studies in Chapter 4, we give a brief overview now for clarity. Due to the difficulty of data acquisition in clinical studies, both datasets are rather small. The data used in [2] (referred to as the *Brooklyn dataset*) consists of coughs collected from confirmed TB-positive patients and from healthy controls (people who were not sick). Recordings were collected in a noise-isolated recording booth with a field recorder, at a sampling rate of 48kHz (later down-sampled to 44.1kHz for feature extraction). In total, coughs from 38 people were collected resulting in 6.23 minutes of coughs audio. In contrast, the dataset used in [3] (referred to as the *Wallacedene dataset*) was collected at a primary healthcare clinic, and all participants were suspected of having TB at the time of recording. The definitive diagnosis for each patient was determined subsequently by laboratory sputum analysis. Recordings were made in a booth with minimal noise isolation adjacent to a busy road, and captured background noises such as chatter, vehicle engines, music and traffic. The audio was recorded with a field recorder with a sample rate of 48kHz, which was later down-sampled to 44.1kHz for feature extraction. In total there were 51 participants in the study, yielding 10.27 minutes of coughs.

2.3.2. Experimental setup

We now detail the experimental setup used in both studies, which includes the classification procedure, feature extraction and experimental method.

Classification

Both [2] and [17] investigated several different shallow classifiers. These included LR, hidden Markov models (HMMs), SVMs, KNN and decision trees. In addition to the aforementioned, FNNs, LSTMs and CNNs were investigated in the second study however their application was largely unsuccessful. For those classifiers that had adjustable architectural hyper-parameters, different configurations were explored through a computationally expensive grid search. Due to the simple nature of these classifiers (with the exception of the deep networks explored in [3]), sequences of acoustic feature vectors (such as those extracted when computing MFCCs) could not be directly processed by the models. Instead, the classification of a patient's single cough was performed by first segmenting it into N segments, and then classifying the time-averaged acoustic feature vector of each segment, disregarding any temporal information. Then, the computed probabilities for all segments for each cough were averaged to yield a *TB index score* (TBI_1), which represents the probability of a patient having TB. The equation to compute TBI_1 is shown below, where N_1 is the number of segments across all acoustic feature representations of all the coughs for a given patient and $\boldsymbol{x} \in \mathbb{R}^d$ is the acoustic feature vector for a single segment (where d is the dimensionality of the acoustic feature vector). It should be noted that [3] in fact used two scores to perform classification. The additional metric (TBI_2) is described in Equation 2.4, where N_2 is the total number of coughs originating from a given patient, and C is the binary TB label determined by comparing the mean frame-wise predicted probabilities to decision threshold γ.

$$TBI_1 = \frac{1}{N_1} \sum_{i=1}^{N_1} P_{\text{TB}}(\boldsymbol{x}_i) \tag{2.3}$$

$$TBI_2 = \frac{1}{N_2} \sum_{i=1}^{N_2} C \tag{2.4}$$

The final prediction of a patient's TB status is made by utilising the index score(s) as shown below, where γ is a decision threshold determined while developing the model.

$$TB = \begin{cases} 1 & TBI_1 \geq \gamma \text{ (used in [2] and [3])} \\ 1 & TBI_2 \geq 0.5 \text{ (used in [3])} \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

Feature extraction

In both studies, only two acoustic feature vectors are investigated, namely MFCCs and linear filter bank (LFB) energies. The particulars of these two acoustic feature representations are detailed in later chapters, but both attempt to emphasise important frequency information whilst reducing dimensionality, with the former being specifically designed for speech-processing applications. Both rely on the short-time Fourier transform (STFT) and as such both the frame length and hop length are key hyper-parameters to optimise during development. Increasing the former increases frequency resolution at the expense of time resolution, while reducing the latter improves time resolution. The relationship between the two is investigated in both studies, with frame and hop lengths ranging from 256 and 4096. The authors also optimise the dimensionality of each acoustic feature vector by varying the number of filters for LFB energies or the number of cepstral coefficients for MFCC's.

The number of segments N into which a cough was divided was also investigated. N

ranged from 1 (simply representing the mean of the entire acoustic feature vector sequence of a given cough) to 4. This was motivated by work presented in [1], which showed cough can be divided into distinct regions, representing different phases of the coughing sound.

Acoustic features were further optimised using SFS, which as briefly mentioned in the preceding section, performs a greedy search to find an optimal subset of acoustic features. This is accomplished by initialising this subset to be empty and sequentially adding the single feature that results in the largest increase in performance. This is determined by training and evaluating classifiers for each combination of the existing subset and the remaining candidate features. This process allows for the discovery and retention of important features for classification, whilst discarding features that result in decreased performance. By simplifying the feature space and subsequent model complexity, substantial performance improvements in classification for low-resource tasks are often achieved. In addition, one can analyse the features selected to be most important and infer previously unknown trends in the data. For example, it is very useful to know the range of frequencies deemed to be important for TB cough classification, since the distinction between a cough that is either TB positive or negative is believed to be impossible with the human ear.

Development and testing

The experimental setup used in both studies is identical. Due to the small size of each dataset, and the desire to present more robust results, a nested k-fold cross-validation strategy is used to both develop and test models. The outer fold is responsible for train/test splits, whilst an inner fold further splits the respective training set into other train/development sets. Each outer loop's inner folds are used to independently select model hyper-parameters and decision thresholds. The exact setup used is visualised in Figure 2.2. The outer loop is used to produce test scores for each classifier, whilst inner loop A is responsible for automatically finding optimal model hyper-parameters, and inner loop B, which has a larger partition, is responsible for determining the EER which is subsequently used to select the optimal decision threshold γ.

2.3.3. Results

Although [2] also presents results on models that utilise both cough and clinical data to distinguish between TB patients and healthy controls, the focus of this study is solely on cough classification and as such we only present the results of classifiers that utilised cough alone. In addition, [2] only discusses results for their best classifier, LR, as presented in Table 2.2. We note that the study finds that MFCCs perform significantly worse than LFB energies, with relative decreases of 35.0%, 13.7%, 21.3% and 12.3% for sensitivity, specificity, accuracy and AUC respectively. Performing SFS on the LFB energies improved

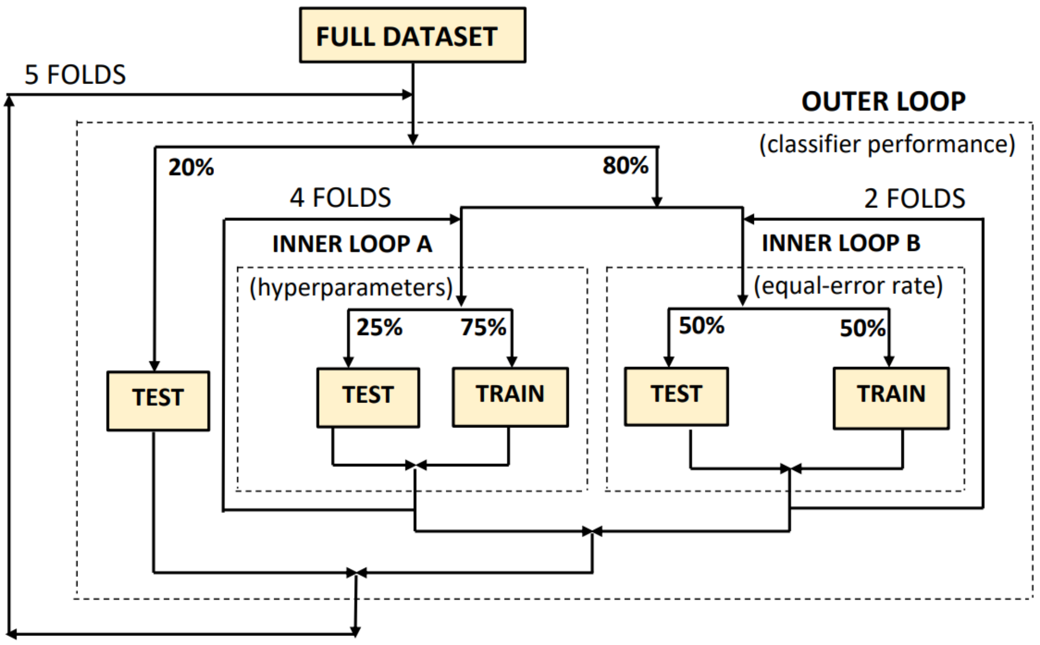


Figure 2.2: The nested k-fold cross-validation strategy used to develop and test classifiers in [2] and [3]. Reproduced from [3].

Table 2.2: Test set classification performance as presented in [2].

Model	Feature	Sensitivity	Specificity	Accuracy	AUC
LR	MFCC	0.40	0.82	0.63	0.71
LR	LFB energies	0.62	0.95	0.80	0.81
LR	LFB energies (SFS)	0.60	0.78	0.78	0.95

AUC significantly (from 0.81 to 0.95), whilst deterioration was observed in the other metrics - indicating that the decision threshold was non-ideal.

In [3] results are presented for several of models and feature hyperparameters. Here, we only consider the results obtained with their best model (LR) and the corresponding optimal features (26 MFCCs with velocity and acceleration). We also include the results for the same model, but using LFB energies as presented in [2]. Unfortunately, results using the exact configuration in [2] (140 filter banks) were not presented in [3], so we present the results obtained with 60 filter banks. These are tabulated in Table 2.3. We note that in contradiction to [2], MFCCs outperform LFB energies. The authors hypothesise that the reason for this improvement was due to the incorporation of more cepstral coefficients, increasing the acoustic feature vectors dimensionality, in addition to including coefficients velocity and acceleration. The reported results are very impressive and meet the WHO's requirements for a minimum sensitivity and specificity of 0.9 and 0.7 respectively for community-based triage testing comfortably [39].

Table 2.3: Test set classification performance as presented in [3].

Model	Feature	Sensitivity	Specificity	Accuracy	AUC
LR	LFB energies	-	-	-	0.75
LR	MFCC	-	-	-	0.86
LR	MFCC (SFS)	0.93	0.95	-	0.94

Both studies demonstrated the nuance of the frequency information being learnt for classification. In [3], it was concluded that models were relying on frequency information that is typically not resolvable by the human ear (such as frequency bins too closely spaced together). This was shown by observing that the higher resolution MFFC features (26 or 39 cepstral coefficients), which were not investigated in [2], performed substantially better than a lower resolution baseline (13 cepstral coefficients). The discrepancy in performance indicates that the model was relying on the increased frequency resolution afforded by the former. In [2], by applying SFS to LFB energies and analysing the subsequently determined optimal features, frequency ranges important for classification were inferred by inspecting each features corresponding filter banks centre frequency. In particular, the most important features corresponded to filters with centre frequencies at 48Hz, 79Hz, 236Hz, 550Hz, and 10418Hz. In agreement with [3], it was concluded that some of these frequencies were spaced too closely together to be discernible by the human ear.

2.4. General acoustic analysis

Cough classification is a highly under-researched field and as such, the literature is quickly exhausted when searching for potential deep learning architectural inspiration. However, it falls under the general umbrella of acoustic analysis, and therefore it may be reasonable to explore seemingly adjacent domains that are more well-researched. There are many fields that fulfil this criterion (emotion recognition, music synthesis, environmental noise classification etc.). We choose to provide brief descriptions of notable work in the areas of acoustic scene classification, keyword spotting (KWS) and automatic speech recognition (ASR) for potential inspiration in neural network model architectures (and pre-trained models) that may be relevant to cough classification.

2.4.1. Acoustic scene classification

Typical current approaches to acoustic scene classification rely on end-to-end CNN-based architectures which learn mappings directly from audio spectrogram inputs. These architectures have performed well due to their inherent ability to learn hierarchical spatial information which is well suited to the non-linear frequency and temporal characteristics of environmental sounds [40, 41]. Since we have already highlighted the application of such architectures to cough classification, we turn our focus to a more recent development in the field - the widespread use of transformers.

Whilst we detail the transformer architecture in thorough detail in Chapter 3, in essence, it is a non-autoregressive sequence model, and consequently allows for much larger architectures to be trained in substantially less time compared to similar-sized auto-regressive counter-parts such as recurrent neural networks (RNNs). The authors of [5] propose a convolution-free audio spectrogram transformer (AST) model, which is based on the vision transformer architecture (ViT) [42]. Patches of a spectrogram are encoded via a linear projection which is subsequently arranged as a sequence and fed into a transformer encoder. By including a positional encoding which contains information regarding the original location of the patch in the input spectrogram, and by means of the self-attention performed by the transformer, the architecture is able to learn context-rich latent image representations suitable for classification. The authors show that the proposed architecture improves upon all previous state of the art CNN based solutions on datasets such as AudioSet [43], ESC-50 [35] and Speech Commands [44].

Large transformers such as AST benefit from lots of data, consequently, training such a large transformer from scratch would not be feasible with the small TB datasets that currently exist. With the success achieved when fine-tuning large pre-trained out-of-domain architectures for COVID-19 classification (in particular ResNet), similar success may be achieved when fine-tuning AST (which has already learned general audio representations) for cough classification.

2.4.2. Keyword spotting

As opposed to traditional ASR, whereby the transcription of an entire utterance is proposed (which is generally a computationally expensive process), keyword spotters simply indicate whether specified keywords are present in an utterance or not. This simplification allows for KWS models to parameter count to be significantly smaller than their ASR counterparts, which in turn means a lot less training data is needed for such models to converge to a local minima, and an overall smaller computational footprint.

One such light-weight and previous state-of-the-art architecture is a recurrent neural network which combines a CNNs ability to capture local information, an RNN's ability to capture long-term time dependencies in the audio data, and an attention mechanism (further detailed in Chapter 3) to weight the RNN's outputs according to their temporal relevance. Specifically, 2D convolutions are performed only along the time axis (by constraining the kernel to have size 1 along the frequency dimension) to extract short-time local relations. This is then fed into a set of bidirectional long short-term memory units (BiLSTMs) to extract long-term dependencies. Finally, this sequence is fed to an attention head which outputs a weighted average by attention score of the BiLSTM output sequence (whereby the key is arbitrarily the vector from the centre of this sequence) which is finally fed to a fully connected network for classification. By combining these three building blocks, the authors were able to achieve state-of-the-art results on Speech Commands (at the time of publication), which was further surpassed by introducing the concept of multi-head attention into the model [45, 46]. Additionally, through analysing the temporal weightings determined by the attention mechanism, the authors were able to visualise the location of the keyword in utterance.

The simple combination of neural network building blocks to achieve substantial improvement compared to the previous best methods is important to consider when developing our own architectures. Additionally, the use of the attention mechanism to objectively inspect the temporal regions learnt to be important may be useful when understanding the signal being learnt by TB classifiers.

2.4.3. Automatic speech recognition

ASR is a well-researched field. This is partly due to the abundance of training data available for high-resource languages such as English, where it is not uncommon for models to be trained on hundreds of hours of transcribed audio. Whilst traditional ASR techniques have broken up the process of performing ASR into multiple stages (feature extraction, acoustic modelling, lexicons and language modelling), recent work has focused on fully end-to-end deep learning -based systems, which in some cases perform the ASR task directly from the audio waveform. Such end-to-end systems are typically large ($>$ 100M parameters) and require significant computing power and data to train. However, once these models have been trained, their ability to capture complex latent information allows the transfer of the model to auxiliary tasks (for example, KWS [6]).

In particular, we pay attention to wav2vec2.0, a self-supervised transformer-based framework which receives acoustic representations extracted directly from a speech waveform using a CNN feature extractor. The model is pre-trained on the LibriSpeech corpus, containing 960 hours of audio [47], whereby latent representations of speech are learned by solving a contrastive task. During training, regions of audio are masked and the model is tasked with identifying the true corresponding quantized latent representation (determined by passing the same unmasked sequence through the model) of the masked time-step from a set of distractors (latent representations that do not correspond to the masked time-step). After this self-supervised pre-training, the model is fine-tuned with labelled data by attaching a linear layer on top of the network which projects the latent representations into C classes representing the vocabulary of the given task (in this case individual characters and special tokens for word boundaries, start, end, and blanks), optimised with CTC loss [48]. With just 10 minutes of labelled data and a transformer-based character-level language model, a word error rate of 4.8% and 8.2% was achieved on the Librispeech test clean and other sets respectively. These are reduced to 2% and 4% when trained on 100 hours of labelled data.

Despite cough being characteristically different from speech, it has been shown in previous studies that the important frequency information being learnt lies within the same bands as speech. Moreover, motivated by the success of transfer learning in other cough classification studies, a fine-tuned wav2vec2.0 may be able to extract good acoustic representations for TB cough classification.

2.5. Summary

In this chapter, we reviewed the literature relevant to the objectives of this work. We began with a high-level description of the pathology and transmission of tuberculosis. We discussed research that details the acoustic analysis of cough, and literature that describes cough classification for a variety of respiratory conditions, including wet/dry cough distinction, pneumonia and COVID-19. In each case, the methods and results of each work are presented and discussed, and the specific novelties of each are highlighted. Next, we described in detail previous studies completed on TB cough classification, which showed that simple linear classifiers achieve promising classification performance. Lastly, we described work of interest that was not directly related to cough classification, but rather state-of-the-art applications of machine learning to other acoustic tasks. These will serve as inspiration for architectures explored in this work which will be detailed in subsequent chapters. In the next chapter, we will provide background and descriptions of general deep-learning building blocks and techniques, as well as describe the acoustic feature vectors used in this work.

Chapter 3

Background

Whilst previous work in TB cough classification has successfully shown that shallow classifiers can perform well, similar performance with the use of deep learning architectures has eluded researchers. Due to the increased size of the dataset used in this work, we place our focus on using deep architectures that have seen widespread success in the broader machine learning field. In this chapter we describe architecture *building blocks* used in this study, namely: feed-forward neural networks (FNN), convolutional neural networks (CNN), recurrent neural networks (RNN), attention and transformers. This is then followed by a summary of basic principles used when training deep neural networks including optimisation, loss functions, and techniques such as dropout and batch normalisation. Lastly, we introduce three prominent acoustic feature vectors that will experimented with in this work. Namely, linear filter bank (LFB) energies, mel-spectrograms and mel-frequency cepstral coefficients (MFCCs).

3.1. Feed-forward Neural Network

The feed-forward neural network (FNN), also referred to as multi-layer perceptron or fully connected network, forms the foundation of deep learning. The goal of any classifier is to best approximate a target output distribution $\boldsymbol{y}$ by defining a mapping $\hat{\boldsymbol{y}} = f(\boldsymbol{x};\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ are learnt parameters that result in the best mapping. In the case of a FNN, this mapping is derived by stacking functions such that the output of one becomes the input to the next: $f(\boldsymbol{x}) = f^{(n)}(f^{(n-1)}(...f^{(1)}(\boldsymbol{x})))$. Each function is described as a layer in the network, where $f^{(1)}$ is the *input layer* as it directly *sees* the input $\boldsymbol{x}$, whilst the final layer $f^{(n)}$ is described as the *output layer*. The layers between the input and output layers are not directly exposed to the training sample, and are therefore called *hidden layers* [49].

Each one of these layers consists of a number of *neurons*, where each neuron receives a vector input which is the concatenation of the scalar outputs from all the neurons in the previous layer, from which it computes a scalar *activation*, similar to linear regression. However, unlike linear regression which simply fits a linear function to its input, each neuron applies a linear function to a non-linear transformation of its input transformed with some non-linear activation function ϕ (non-linear functions are described in more detail in later sections in this chapter). The inclusion of the activation function allows FNNs to fit non-linear functions which capture more complex relationships. To cement this explanation, we visualise this operation for a single neuron, and then an entire layer in an FNN. Given the output of the previous layer $\boldsymbol{h}^{(l-1)} \in \mathbb{R}^{M_{l-1}}$ where M_{l-1} is the number of neurons in the previous layer, the scalar output of m^{th} neuron in the next layer l is

$$h_m^{(l)} = f_m^{(l)}(\boldsymbol{h}^{(l-1)}) = \phi(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{h}^{(l-1)} + b) \tag{3.1}$$

Where $\boldsymbol{w} \in \mathbb{R}^{M_{l-1}}$ and b are learnable weights and bias used to map $\boldsymbol{h}^{(l-1)}$ to the non-linear output. For all the neurons in a given layer in an FNN, this operation can be realised with a single matrix multiplication, where the vector output of the layer is described as $\boldsymbol{h}^{(l)} = f^{(l)}(\boldsymbol{h}^{(l-1)}; \boldsymbol{W}, \boldsymbol{b})$. The mapping function $f^{(l)}$ is described in Equation 3.2 for a layer with M_l neurons where $\boldsymbol{W} \in \mathbb{R}^{M_l \times M_{l-1}}$ are the weights of the linear transformation and $\boldsymbol{b} \in \mathbb{R}^{M_{l-1}}$ are the biases [49].

$$\boldsymbol{h}^{(l)} = f^{(l)}(\boldsymbol{h}^{(l-1)}) = \phi(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}) \tag{3.2}$$

Intuitively we can see how this is the concatenation of each neurons scalar activation. The layered structure is further reinforced by visualising a simple FNN in Figure 3.1. The network consists of an input layer with three neurons which takes an $\boldsymbol{x} \in \mathbb{R}^n$, this feeds into a single hidden layer of with two neurons, which subsequently feeds into the output layer which consists of a single neuron.

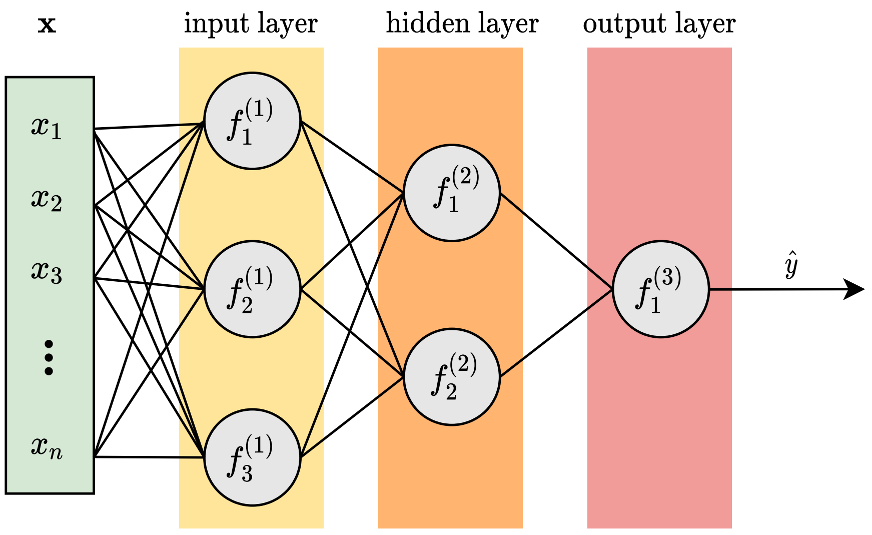


Figure 3.1: A simple example of a three layered feed-forward network consisting of an input layer (yellow), a hidden layer (orange) and a output layer (red). Geen highlights the input feature vector which is fed into the first layer.

3.2. Convolutional Neural Network

CNNs are able to extract features in high dimensional spaces using series of low-level filters which map this information to new latent spaces. Performing this operation enough times allows for highly complex representations to be learned. This process hinges on the assumption that the input to the network is image-like, and therefore can constrain the process by which these latent features are extracted. In particular, information is not necessarily position dependent and therefore the network should be able to detect it independently from where it appears. The process by which this accomplished is described subsequently.

3.2.1. Convolutional layers

Concretely, a convolutional layer utilises filters or *kernels* that are *convolved* with its input. Typically these kernels are much smaller than the input, and the convolution operation involves moving them over an image in a pre-determined manner. Before providing a mathematical explanation, we first visualise this operation in Figure 3.2, whereby a single channel input is convolved with one 2×2 kernel. The kernel is moved across the input with a stride of one i.e. it can either move horizontally or vertically by one element when traversing the input, whereby at each step the aforementioned element-wise product between the kernel and corresponding overlapped region of the input is computed and summed - resulting in the activation for those specific coordinates in the input. Through this process, the network introduces the concept of *sparse connectivity* and *parameter sharing*. Unlike FNNs, where every neuron from the previous layer is used to compute the activation of every other neuron in the next layer, each output unit of a convolutional layer is only only connected to relatively few input units. These *sparse connections* mean kernels can focus on small meaningful features such as edges in images (or formants in spectrograms), whilst reducing the number of computations. Since each a kernel activation is computed at almost every position in the input, CNNs inherit a property called *invariance to translation* and parameters of the kernel are shared across the input. That is, if some bit of information were to appear somewhere unexpected in the input, the same kernel would be still be able to detect it, and the information would move a corresponding amount in the output feature space. This is unlike a FNN, which would have to re-learn the same feature at each possible input location. This phenomena is visualised in Figure 3.3, whereby a hand crafted 3×3 kernel (a Prewitt edge detection filter) is convolved with the input, irrespective of where edges may appear in the input image, they are represented in the activation map.

In practice, a single CNN layer will have a multi-channel input (for instance the three channels of an RGB image), and therefore is not typically two dimensional, but rather

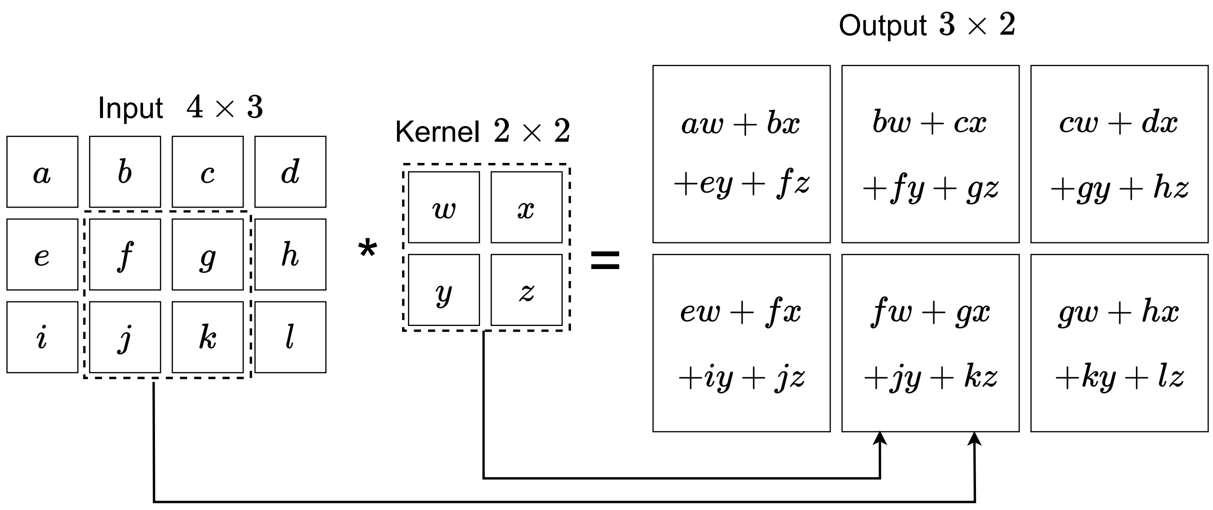


Figure 3.2: The output of the 2-D convolution between a 3×4 input and a 2×2 kernel. Each time the kernel is stepped across the input the element-wise product between the kernel and corresponding overlapped region of the input is computed and summed.

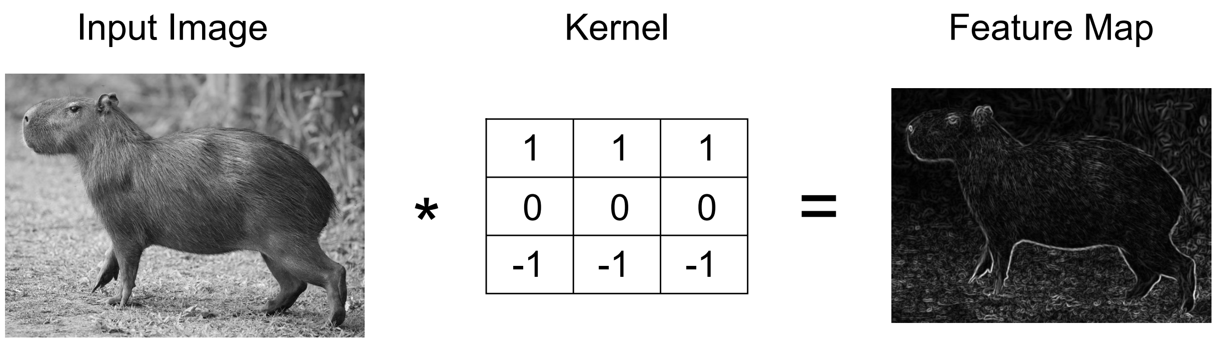


Figure 3.3: A 3×3 edge detection kernel is applied to an image of a capybara.

a three dimensional tensor, with dimensions $C_{\text{in}} \times W_{\text{in}} \times H_{\text{in}}$, where C_{in} is the number of input channels, W_{in} is the width, and H_{in} is the height. In this case, kernels have also have a third dimension equal to C_{in}. Additionally, a CNN layer would typically have and multiple kernels. The output of each kernels convolution however is still a two-dimensional activation map ($W_{\text{out}} \times H_{\text{out}}$), which is then stacked along a new channel dimension together with the other kernels output feature maps from the same layer. As such, the number of output channels C_{out} of a CNN layer is equivalent to the number of kernels in that layer.

Mathematically, the convolution is not actually computed, but rather the cross-correlation. The computation for a single scalar activation between a kernel and overlaping region in the input is described in Equation 3.3, where $\boldsymbol{X} \in \mathbb{R}^{W_{\text{in}} \times H_{\text{in}}}$ is the two dimensional input and $\boldsymbol{K} \in \mathbb{R}^{M \times N}$ is the kernel, (i, j) is the anchor point of the region of overlap in the input (top left corner), and M and N are the dimensions of the kernel. As previously indicated, this operation is visualised in Figure 3.2.

$$S(i,j) = (\boldsymbol{X} * \boldsymbol{K})(i,j) = \sum_{m}^{M}\sum_{n}^{N} \boldsymbol{X}(i+m, j+n)\boldsymbol{K}(m,n) \tag{3.3}$$

3.2.2. Stride and padding

Aside from the number of filters, there are two other hyper-parmeters that effect the dimensonality of the activation map. Although we have mentioned stride previously, we will subsequently describe it together with padding in detail.

- **Stride** refers to the step size (both horizontal and vertical) with which a kernel is moved across the input to which it is being convolved. A stride of one ensures that a kernel is applied to as much of the input as possible, preserving the granularity of information. However, this can result in large activation maps, and be computationally expensive. In this case, a larger stride is used.

- **Padding** is a method to prevent dimensionality reduction when applying convolution to an input. This is accomplished by artificially increasing the number of convolution operations that take place (and therefore the size of the activation map) by padding a boarder of zeros with a specific width around the input.

The effects of both the stride (S) and amount of zero padding (P) on the output dimensions W_{out} and H_{out} with respect to the inputs dimensions W_{in} and H_{in} are described in Equations 3.4 and 3.5 respectively (recall M and N are the width and height of the kernel).

$$W_{\text{out}} = \frac{W_{\text{in}} - M + 2P}{S} + 1 \tag{3.4}$$

$$H_{\text{out}} = \frac{H_{\text{in}} - N + 2P}{S} + 1 \tag{3.5}$$

3.2.3. Convolutional block

Typically, the convolutional layer consists of 2 or 3 stages: convolution with the kernels, a non-linearity and an optional *pooling* operation. Pooling replaces the output at a certain location with a summary statistic of neighbouring outputs. For example, in *max pooling* the output is reported as the maximum output in a square neighbourhood around it. This typically makes the model more robust since if a small translation occurs, the values of the pooled outputs mostly stay the same. However, this comes at the cost of losing the ability to precisely know where a particular feature is (which in most cases is not of particular importance). In large architectures, a pooling layer may only be used after a stack of convolutional layers to prohibit latent feature maps from being shrunk too small. The structure of a typical CNN layer is presented in Figure 3.4, whereby we denote the output of the block as $\boldsymbol{S}' \in \mathbb{R}^{C'_{\text{out}} \times W'_{\text{out}} \times H'_{\text{out}}}$.

Once a sufficient number of convolutional blocks have been stacked, the extracted three-dimensional latent feature representations are typically used for a task, for example, image classification. This requires the use a fully connected layer, which takes a vector as input. A naive method is to simply flatten feature representation into a vector of shape $C \cdot W \cdot H$. Another popular technique is to apply a channel pooling layer, whereby a single 1×1 kernel is used to pool all channels and hence reduce the feature representation's complexity before flattening, resulting in a lower dimensional $W \cdot H$ vector.

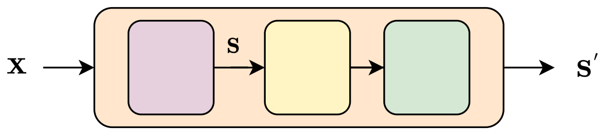


Figure 3.4: Typical structure of a convolutional block, consisting of a convolutional layer, non-linear activation function, and an optional pooling layer.

3.2.4. Residual connections

When training very deep networks (networks with many layers) gradients tend to vanish as the loss is back-propagated to earlier layers. For CNNs a widely used mitigation are residual connections between groups of layers, which act *highways* for the loss to back-propagate to earlier layers in the network (parameter updating through back-propagation is explained in more detail in subsequent sections). A residual connection (or skip connection) for a set of convolutional blocks is depicted in Figure 3.5 whereby the output of the residual block is $f(\boldsymbol{X}) + \boldsymbol{X}$, where $f(.)$ is the function that describes the operations performed within the residual block itself.

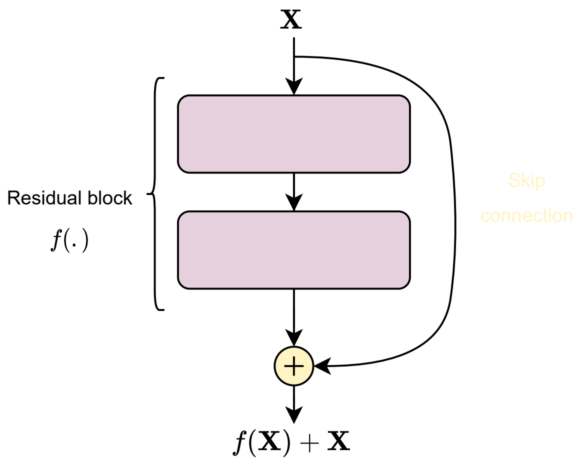
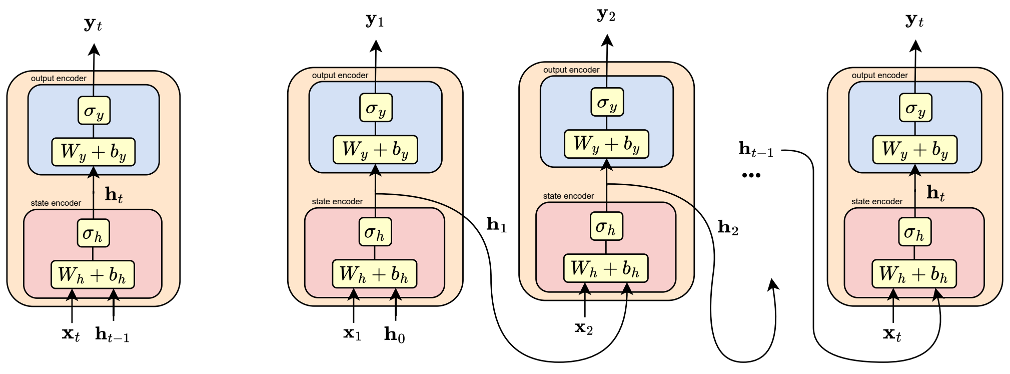


Figure 3.5: Depiction of a residual block and its skip-connection. The input to the block is summed with the output.



(a) RNN cell.

(b) Unfolded RNN in time.

Figure 3.6: A single RNN cell is shown, as well as an unfolded RNN in time showing the parsing of state information from previous time steps.

3.3. Long Short-Term Memory Networks

RNNs offer a powerful utility in modelling time dependencies in sequences and have become baseline systems in many speech-related tasks (voice activity detection, automatic speech recognition and keyword spotting). We briefly provide a general overview of RNNs and then describe the LSTM cell which addresses some of the challenges faced by the traditional RNN architecture.

3.3.1. Recurrent Neural Networks

Traditional neural networks do not have the ability to retain information from previous inputs, which is problematic when dealing with a sequence of inputs whereby the temporal axis contains important information, such as an acoustic signal. To circumvent this, an RNN builds upon feed-forward networks by actively passing information about the previously seen inputs in a sequence forward in time. This *feedback connection* allows information to flow from one time step to another by passing a state representation from the previous time step to the hidden layers of the network and thus allowing the network to leverage previous information in time for current computations.

The structure of an RNN is shown in Figure 3.6, with the single RNN cell structure on the left, and high-level representation of the recurrent processing of inputs and sharing of information between states shown on the right. The RNN cell has two sets of trainable weights ($\boldsymbol{W}_h$ and $\boldsymbol{W}_y$) and biases ($\boldsymbol{b}_h$ and $\boldsymbol{b}_y$) and two non-linear activation functions (σ_h and σ_y). Note we do not give the dimensions of these weights, as they are symbolic, and in fact, represent a set of weights that are used in a sequence of operations to produce the desired output. The first half the of the cell is responsible for encoding the *hidden state* $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ with size d_h which includes information about previous time steps $\boldsymbol{h}_{t-1}$ and the current feature vector $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$ with size d_x. The second half turns $\boldsymbol{h}_t$ into a output

$\boldsymbol{y}_t \in \mathbb{R}^{d_y}$ with size d_y suitable for the desired application. This update rule is represented by the following equation:

$$\begin{aligned} \boldsymbol{h}_t &= \sigma_h(\boldsymbol{W}_h(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_h) \\ \boldsymbol{y}_t &= \sigma_y(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y) \end{aligned} \tag{3.6}$$

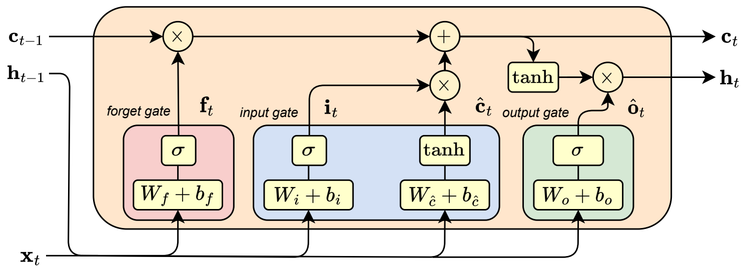
RNNs, like most modern neural networks, are trained using back-propagation with the caveat that the error derivatives must be back-propagated through all considered instances in time (back-propagation through time). In doing so, gradients are repeatedly multiplied through time, which can lead to exploding/vanishing gradients for long sequences [50]. This makes it hard to learn long-term dependencies and achieve stability during training. Due to this, in practice, the RNN shown in Figure 3.6 is not a viable architecture. Alternatives that mitigate the exploding and vanishing gradients have been proposed, the most notable being the LSTM network.

3.3.2. LSTM network

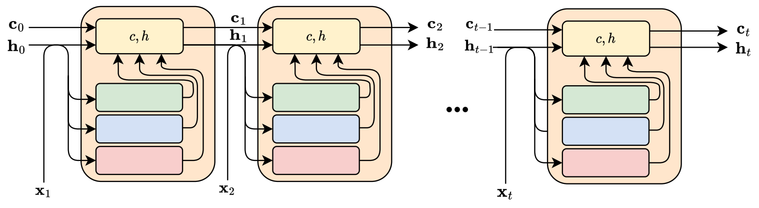
The LSTM addresses the aforementioned vanishing/exploding gradient problem by introducing a series of information control structures: A cell state $\boldsymbol{c} \in \mathbb{R}^{d_h}$, as well as three *gates* which introduce further control with regards to the flow of information in time. Namely: the forget gate, input gate and output gate. A diagram of the cell structure is shown in Figure 3.7a, and a high-level representation of an unfolded LSTM network in time in Figure 3.7b. Note the sigmoid activation function and hyperbolic tangent function are denoted by σ and *tanh* respectively. By simply observing the structure of the cell, we can see that the network solves the vanishing gradient problem by providing a direct path from cell state $\boldsymbol{c}_t$ to $\boldsymbol{c}_{t-1}$. Gradients can flow along this path without exponentially decaying/exploding [50]. Each component is described in detail below. We omit the dimensions of each respective weight matrix as with the RNN as they are symbolic, and in fact, represent a set of weights that are used to produce the desired output. For example, aggregating information between $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ in each gate is non-trivial due to potential dimension mismatch and requires a separate weight matrix for each to accommodate this.

Forget gate: The forget fate activation vector $\boldsymbol{f}_t \in \mathbb{R}^{d_h}$ is responsible for determining which parts of the previous cell state $\boldsymbol{c}_{t-1}$ are kept and which parts are forgotten. Thus, the forget gate determines which information from previous time steps will remain encoded.

Input gate: The input gate is responsible for computing the input gate activation vector $\boldsymbol{i}_t \in \mathbb{R}^{d_h}$ and candidate cell state $\hat{\boldsymbol{c}}_t$. The vector, $\boldsymbol{i}_t$ dictates which parts of $\hat{\boldsymbol{c}}_t$ will be used when computing the new cell state in a similar fashion to the forget gate.



(a) Structure of a single LSTM cell.



(b) High level representation of an LSTM unfolded in time.

Figure 3.7: A single LSTM cell, showing in detail the computations performed with each gate's (where red, blue, and green highlight the forget, input, and output gates respectively) output and the cell state. In addition, a high level representation of an LSTM unfolded in time.

Cell state: The updated cell state is a combination of information from the previous state that is to be maintained and information from the input gate that is to be maintained. This is best described by the equation below, whereby $\circ$ denotes the element-wise product.

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \hat{\boldsymbol{c}}_t \tag{3.7}$$

Here $\boldsymbol{f}_t$ controls the information flow from the previous time step, and $\boldsymbol{i}_t$ controls the information flow from the current time step.

Output gate: The output gate activation vector $\boldsymbol{o}_t \in \mathbb{R}^{d_h}$ is used to suppress unwanted information in the hidden cell state $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$, and determines the output of the network (similar to $\boldsymbol{y}_t$ in the classic RNN). The updated hidden state $\boldsymbol{h}_t$ which will be passed to the next time step is determined with the equation below.

$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ tanh(\boldsymbol{c}_t) \tag{3.8}$$

Bidirectional LSTM

A popular adaptation of the LSTM is to process the input in both directions, referred to as a bidirectional LSTM (BiLSTM). In doing so, the network is able to produce latent representations for a specific time-step that captures both past (backward in time) and future (forward in time) information from elements of the sequence. The ability to represent both past and future relevant information for a given element in a sequence is immensely useful when modelling non-causal relationships. For example in natural language processing, the likelihood of a word being spoken can drastically change based on what is said next, and therefore it makes sense to encode this *future* information in its representation. We present a high-level depiction of a unfolded in-time BiLSTM in Figure 3.8. $\overrightarrow{\boldsymbol{h}}$ and $\overleftarrow{\boldsymbol{h}}$ denote hidden states generated for the forward and backward LSTMs respectively. Note how $\overrightarrow{\boldsymbol{h}_t}$ is only dependent on inputs and their representations prior to t and $\overleftarrow{\boldsymbol{h}_t}$ is only dependent on inputs and their representations after t. The concatenation of these two vectors results in $\boldsymbol{h} \in \mathbb{R}^{2 \cdot d_h}$, a latent representation for time t that captures both relevant past and future information.

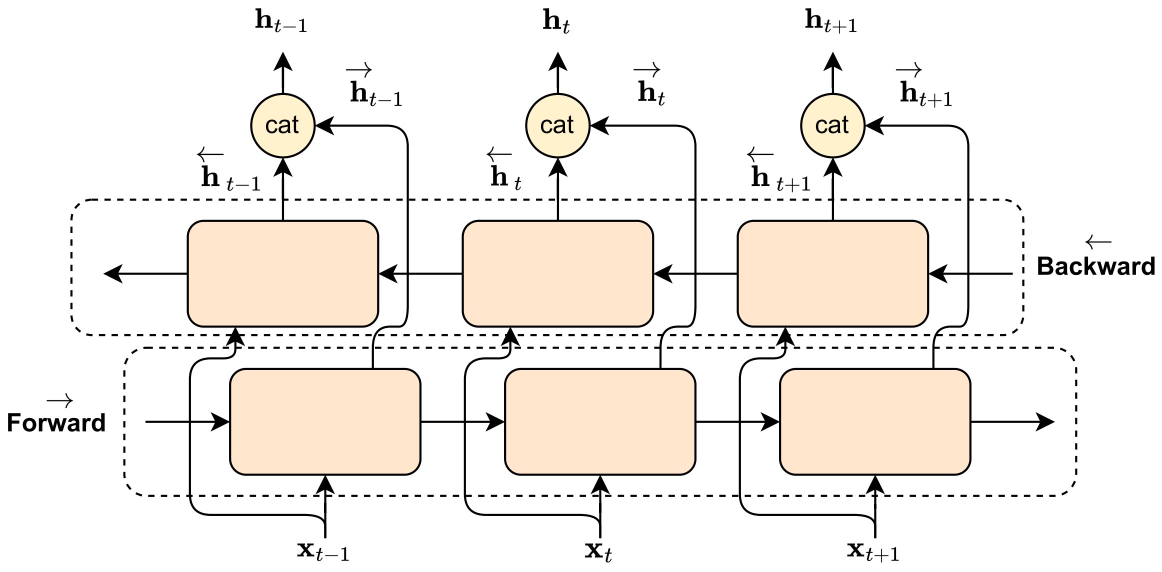

$\mathbf{h}_{t-1}$
$\mathbf{h}_t$
$\mathbf{h}_{t+1}$
$\overrightarrow{\mathbf{h}}_{t-1}$
$\overrightarrow{\mathbf{h}}_t$
$\overrightarrow{\mathbf{h}}_{t+1}$
cat
cat
cat
$\overleftarrow{\mathbf{h}}_{t-1}$
$\overleftarrow{\mathbf{h}}_t$
$\overleftarrow{\mathbf{h}}_{t+1}$
$\overleftarrow{\text{Backward}}$
$\overrightarrow{\text{Forward}}$
$\mathbf{x}_{t-1}$
$\mathbf{x}_t$
$\mathbf{x}_{t+1}$

Figure 3.8: BiLSTM unfolded in time.

3.4. Attention

Attention networks have brought about a radical change in machine learning, especially in language and speech processing [51]. To grasp the underlying concept of attention, a practical example is given first, followed by formal equations that describe the scaled dot-product attention computation [51] - the most prevalent form of attention.

Given some sequence $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t\}$ where $\boldsymbol{x} \in \mathbb{R}^{d_x}$, attention aims to produce a hidden state $\boldsymbol{h} \in \mathbb{R}^{d_h}$, by means of a linear mapping of $\boldsymbol{X}$ with scalar attention weights $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_t\}$ with the constraint that these weights sum to one (soft attention) for each element in a set of query vectors $\boldsymbol{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_m\}$ where $\boldsymbol{q} \in \mathbb{R}^{d_k}$. This operation is described for a single query vector in Equation 3.9. In doing so, the network is able to select information from the sequence most important conditioned on the query and disregard information that is irrelevant.

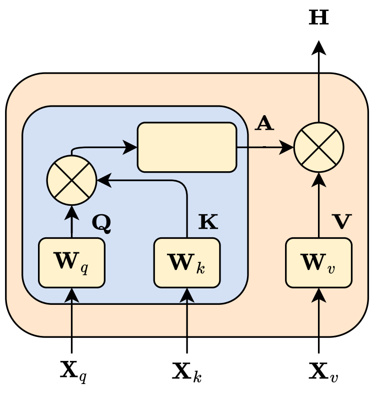


Figure 3.9: Attention cell, where $\boldsymbol{X}_q$ $\boldsymbol{X}_k$ $\boldsymbol{X}_v$ are the set of vectors to be used as queries, keys and values. Note in self-attention $\boldsymbol{X}_q = \boldsymbol{X}_k = \boldsymbol{X}_v$.

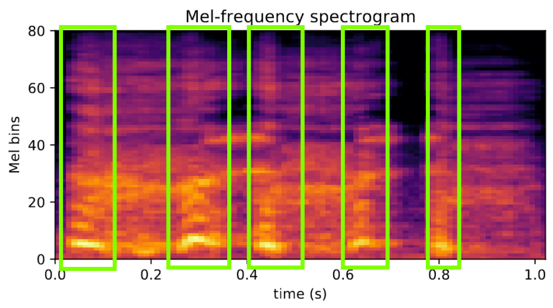
$$\boldsymbol{h}_i|\boldsymbol{q}_i = \alpha_1\boldsymbol{x}_1 + \alpha_2\boldsymbol{x}_2 + ... + \alpha_t\boldsymbol{x}_t \tag{3.9}$$

In the case of self-attention, a hidden state $\boldsymbol{h}_i$ is computed for each sequence element $\boldsymbol{x}_i$ in the input sequence $\boldsymbol{X}$, whereby $\boldsymbol{h}_i$ captures the relevant information from the rest of the input sequence with respect to $\boldsymbol{x_i}$, that is, each element in the sequence is used as a query vector exactly once. The resulting output is a set of hidden states represented by a matrix $\boldsymbol{H} \in \mathbb{R}^{d_h \times t}$ which is computed by performing the dot product between $\boldsymbol{X}$ and $\boldsymbol{A} \in \mathbb{R}^{t \times t}$ the matrix representation of the set of attention weights $\boldsymbol{\alpha}_i \in \mathbb{R}^t$ for each query $\boldsymbol{q}_i$ in $\boldsymbol{Q}$ (which again in the context of self-attention, corresponds to each element $\boldsymbol{x}_i$).

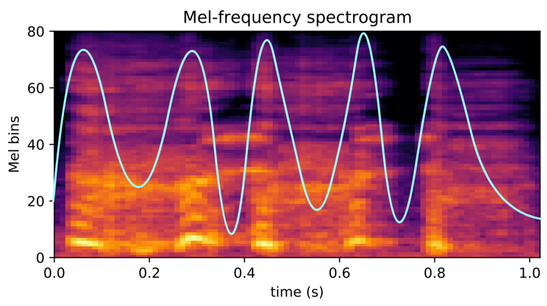
In practice attention is generalised by introducing *key* and *value* sequences $\boldsymbol{K} \in \mathbb{R}^{d_k \times t}$ and $\boldsymbol{V} \in \mathbb{R}^{d_h \times t}$ respectively. The matrix multiplication between $\boldsymbol{Q}$ and $\boldsymbol{K}$ (and softmax of the subsequent output to ensure attention weights sum to one) form the matrix of attention weights $\boldsymbol{A} \in \mathbb{R}^{m \times t}$ which is then multiplied by $\boldsymbol{V}$ to produce $\boldsymbol{H} \in \mathbb{R}^{d_h \times m}$. Note the number of elements in the hidden state matrix is m, the same number of elements as in the query. This operation is described in Equation 3.10 where d_k is used as a scaling factor and is also typically the dimensionality of queries and keys. The attention cell is visualised in Figure 3.9 where $\boldsymbol{W}_q \in \mathbb{R}^{d_k \times d_x}$, $\boldsymbol{W}_k \in \mathbb{R}^{d_k \times d_x}$ and $\boldsymbol{W}_v \in \mathbb{R}^{d_h \times d_x}$ are learnable weight matrices that are applied to the sequences to be used as queries, keys and values ($\boldsymbol{X}_q \in \mathbb{R}^{d_x \times m}$, $\boldsymbol{X}_k \in \mathbb{R}^{d_x \times t}$ and $\boldsymbol{X}_v \in \mathbb{R}^{d_x \times t}$).

$$\boldsymbol{H} = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}^T \tag{3.10}$$

To further cement this concept, we will describe the computation of queries, keys and values in more detail, using a practical single-attention example. Referring to Figure 3.10a, we are presented with a cough spectrogram, of which its time axis can be categorised into three types of audio: silence, unvoiced, and voiced whereby the voiced regions are highlighted in green. Researchers hypothesise that voiced portions of cough contain the majority of the TB signal, and thus want their classifier to weigh the associated temporal



(a) Cough spectrogram with five high energy voiced regions highlighted in green.



(b) Stylised attention weights verses time for the given spectrogram when using a query vector representing the generalised voiced spectra of cough.

Figure 3.10: Visualisation of a practical attention application to cough.

regions more heavily. Consequently, we assume all optimal parameter matrices have already been found by training a system which uses components of the below-described process. We will use the analogy of trying to find all voiced regions in the cough to describe the role of queries and keys when computing the attention weights.

Queries: The query can be interpreted as to what the network computes the attention scores with respect to. In this example, the query is a single vector $\boldsymbol{X}_q \in \mathbb{R}^{d_x \times 1}$ representing a typical spectrum of a voiced frame of coughing audio which we assume we know prior to this search, where d_x is the number of frequency bins. To transform $\boldsymbol{X}_q$ into an actualy query $\boldsymbol{Q}$, we have to multiply our query vector by the learnable weight matrix $\boldsymbol{W}_q$:

$$\boldsymbol{Q} = \boldsymbol{W}_q \boldsymbol{X}_q \tag{3.11}$$

Keys: The keys constitute the set of information which the network is trying to weigh with respect to the query. This would be the sequence of t frames in the spectrogram $\boldsymbol{X}_k \in \mathbb{R}^{d_x \times t}$. Like the queries, the learnable weight matrix $\boldsymbol{W}_k$ is applied to this sequence:

$$\boldsymbol{K} = \boldsymbol{W}_k \boldsymbol{X}_k \tag{3.12}$$

Finally, $\boldsymbol{A} \in \mathbb{R}^{1 \times t}$ is determined as described by multiplying $\boldsymbol{Q}$ and $\boldsymbol{K}$ together, and applying the softmax term as described Equation 3.10. We show the resulting stylised attention scores overlaid with the cough spectrogram in Figure 3.10b. Note the voiced regions have very high attention scores, whilst in the unvoiced regions the attention scores drop towards zero as they blend into silence.

Multi-head attention: Instead of performing single attention with a given set of queries, keys and values, it is often useful to have multiple weight matrices, whereby different hidden states can be learnt for the given sets of queries, keys, and values, and concatenated and projected again to reduce dimensionality. This is known as multi-head attention and allows the model to jointly attend to multiple relationship streams, capturing a more diverse range of contexts. For example, in the application of multi-head self-attention to NLP, one head could focus on attending to relevant nouns in the sentence, whilst another on verbs.

3.5. Transformer encoder

While incorporating attention into a network is useful when one already has highly non-linear latent representations of an input sequence, its simple mapping is not sufficient to capture these complex latent representations on its own. First proposed in [51], the transformer addresses this issue by first incorporating a positional encoding to the input sequence, such that temporal information is retained, and then feeding this sequence into a stack of transformer blocks each containing a combination of self-attention and FNN layers both with residual connections. By stacking a sufficient number, complex relationships can be captured. In the traditional transformer, these context-rich latent encodings are then decoded by transformer decoding blocks, but this step is irrelevant to this study and is used more in traditional NLP tasks such as machine translation.

The transformer encoding scheme is illustrated in Figure 3.11. A sequence $\boldsymbol{X} \in \mathbb{R}^{n\times t}$ is passed through a positional encoder, which simply adds a positional encoding matrix $\boldsymbol{P} \in \mathbb{R}^{d_x\times t}$ to $\boldsymbol{X}$, which in the original work [51] that presents the transformer architecture is derived from an assortment of cosine functions at different frequencies as described by Equation 3.13 where $0 \leq i < \frac{d_x}{2}$ and $0 \leq k < t$. The encoded sequence is then passed to a multi-head self-attention layer, whereby a skip connection and normalisation layer are used to prevent vanishing gradients and stabilise training. The same is repeated for the linear feed-forward layer, whereby the single linear layer is applied to every element in the sequence.

$$\begin{aligned} P(2i,k) &= \sin(\frac{k}{10,000^{2i/n}}) \\ P(2i+1,k) &= \cos(\frac{k}{10,000^{2i/n}}) \end{aligned} \tag{3.13}$$

3.6. Activation functions

An activation function describes how the output of a linear operation is non-linearly transformed. Typically, the activation function used is highly dependent on the goal of the network. Three activation functions are used widely: sigmoid, hyperbolic tan (tanh), and the rectified linear unit (ReLU), and are shown in Figure 3.12 respectively.

Sigmoid and tanh are well-known squashing functions. The former limits the set of all real numbers between $(0, 1)$ whilst the latter between $(-1, 1)$. Whilst these both have strong non-linear properties, a general problem with the sigmoid and tanh functions is that, since they have a finite output range, they become increasingly less sensitive to large values which saturate the functions (values begin to approach the respective functions'

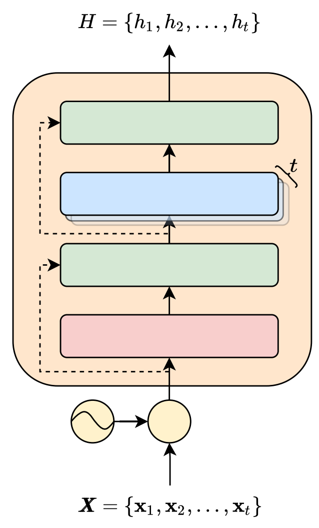

$H = \{h_1, h_2, \dots, h_t\}$
t
$\boldsymbol{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t\}$

Figure 3.11: A single transformer encoding block, whereby $\boldsymbol{X}$ has already been passed through a positional encoding. Dashed lines indicate residual skip connections.

bounds). In the case of deep neural networks, this damages the network's information capacity and learning ability (due to smaller gradients) [52]. ReLU was formulated to address this issue. ReLU behaves linearly for all positive values but outputs zero for all negative inputs. Hence, ReLU maintains the same gradient as inputs grow large and thus retain information sensitivity, but is still nonlinear (which is essential for deep networks to learn complex relationships) due to its piece-wise nature. The ReLU function is described by the equation below.

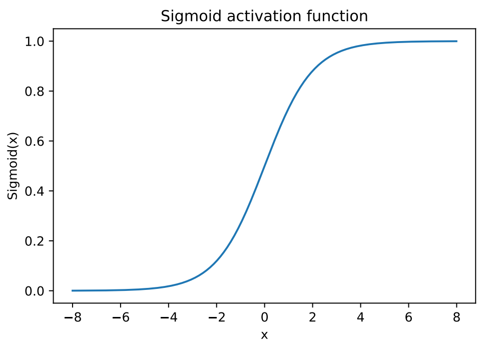
$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \tag{3.14}$$

3.7. Training

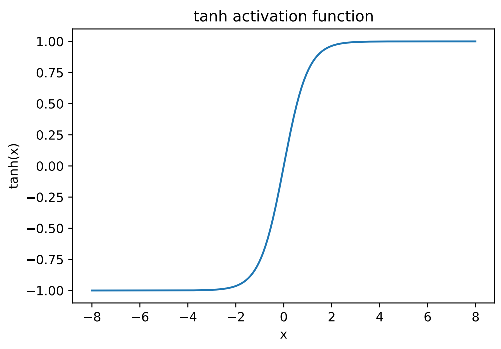
In this section, we describe how a deep neural network is typically *trained*. That is, how optimal parameters are found such that a good solution to the problem is fit. We start by describing the quantification of model error using a loss function. Next, we describe how this loss function is used to iteratively update model parameters such that it is reduced. This is then followed by a series of common techniques (including network structure and training hyper-parameters) that are typically used to improve and optimise the training process.

3.7.1. Loss functions

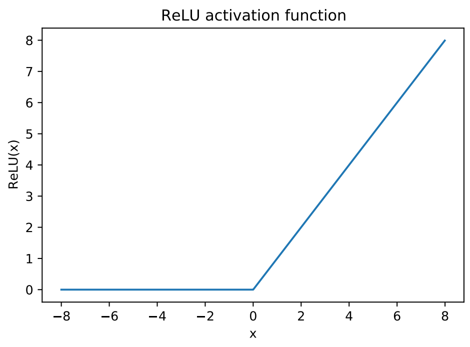
A loss function is used to describe the error made by a model with respect to the desired outcome. For classification, this is usually accomplished by comparing ground truth labels



(a) Sigmoid activation function.



(b) tanh activation function.



(c) ReLU activation function.

Figure 3.12: Commonly used activation functions.

with the predicted label probabilities outputted by a model. Typically, cross-entropy (or equivalently negative log-likelihood) is used. The loss for a single input $\boldsymbol{x}$ is described in Equation 3.15 where $f(\cdot)$ is the model function that outputs a vector that corresponds to predicted probabilities for each class, and $\boldsymbol{y}$ is the ground truth one-hot encoding for c classes, whereby the index of the correct class is assigned one and the rest zero. To ensure the probabilities as predicted by $f(\cdot)$ sum to one, the last layer in the network, which is reasonable for outputting the activation values that correspond to class confidence, is typically followed by a softmax function.

$$J(\boldsymbol{x};\boldsymbol{\theta}) = -\boldsymbol{y} \cdot \log(f(\boldsymbol{x};\boldsymbol{\theta})) \tag{3.15}$$

The relation between model confidence (how large the predicted probability is) and the cross-entropy loss is visualised in Figure 3.13. Clearly, cross-entropy rewards the model for being confident and correct (by means of a low loss), and penalises it for the converse on an exponentially decaying scale. In the case of batched learning (described subsequently), this loss function is summed for each element in a batch and normalised. The resulting value is used by an optimiser to adjust model weights accordingly, the method by which is detailed in subsequent sections. In the case where predictions might be biased to a certain class, for instance in unbalanced datasets, an additional term can be included to artificially increase the underrepresented class' loss, and therefore incentivise the network to place more focus on correctly predicting it. This is aptly named weighted cross-entropy and is practically implemented by introducing an additional term $\boldsymbol{\beta} \in \mathbb{R}^c$ whereby the value of each element corresponds to a respective class's weight. To perform weighted cross-entropy $\boldsymbol{\beta}$ is simply element-wise multiplied by the one-hot encoding $\boldsymbol{y}$.

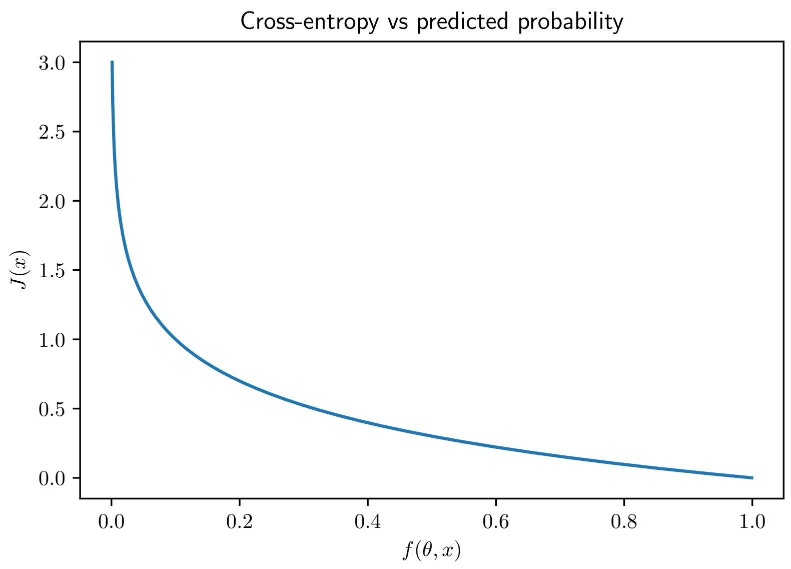


Figure 3.13: Plot of the cross-entropy loss compared to the predicted probability for a given class. As the confidence increases, the loss decays exponentially.

3.7.2. Back-propagation

Back-propagation is a technique used to calculate the gradients of the model parameters $\nabla_{\boldsymbol{\theta}}$ in the network with respect to the loss function J after each iteration. These gradients are then subsequently used to update the model parameters according to the chosen optimisation method. Where:

$$\nabla_{\boldsymbol{\theta}} = \frac{\partial J(f(\boldsymbol{x};\boldsymbol{\theta}), y)}{\partial \theta} \tag{3.16}$$

Whilst the derivation differs depending on the loss function used, the key principle of back-propagation utilises the chain rule to compute each layer's partial derivative. Whereby partial derivatives computed for the n^{th} layer can then be used in the computation of the partial derivatives of the $n-1^{\text{th}}$ layer and so on until all layer's partial derivatives have been computed with respect to the loss function.

3.7.3. Optimisation

As previously eluded to, neural network parameters are typically optimised using some form of gradient-based learning. The goal of gradient-based learning is to iteratively adjust

Algorithm 3.1: Stochastic gradient descent (SGD) parameter update [49]

```
while stopping criterion no met do
    Sample minibatch of $m$ examples from training set
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i J(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
end while
```

the weights in such a way that a chosen loss function approaches a local minima, whereby the gradients will approach zero. In machine learning, stochastic gradient descent (SGD) is a widely used optimisation to achieve this. In each iteration of training, a mini-batch of m samples (sampled from a training set) is used to compute gradient estimates which are subsequently used to update model parameters such that the J approaches a local minima as described in Algorithm 3.1, where ϵ is the *learning rate*.

However, due to the noisy nature of the sampled mini-batches and subsequent gradients, models using this optimisation require more iterations to converge and are sensitive to the learning rate. In this case, a popular technique is the use of *momentum*. Momentum computes a decaying exponential sum of the gradients, mitigating much of the noisy characteristics of the mini-batches and resulting in faster convergence. This is described by Equation 3.17 and 3.18, where $\boldsymbol{v}$ is the *velocity* and $\alpha \in [0,1)$ (the momentum) determines how quickly the contribution of the previous gradients decay:

$$\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i J(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)}) \tag{3.17}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v} \tag{3.18}$$

Whilst SGD and momentum form one of the fundamental optimisers used in deep learning, there exist some more sophisticated approaches that introduce adaptive learning rates on a parameter scale to mitigate axis-aligned sensitivity. Two of these are the Adam optimiser and RMSprop - which will be used in this work. The key difference between the two is that Adam incorporates not only the exponentially decaying average of squared gradients when scaling the learning rate, but also the exponentially decaying average of past gradients. Both are described in more detail in [53] and [54].

Learning rate

The learning rate is used to scale the computed parameter gradients with respect to the loss function and update model parameters accordingly, and as such is a fundamental hyper-parameter in deep learning applications. As illustrated in the simplified example in Figure 3.14, the learning rate can be a very important parameter when attempting to optimally traverse the loss surface toward a local minima. If it is too small, the optimiser will get

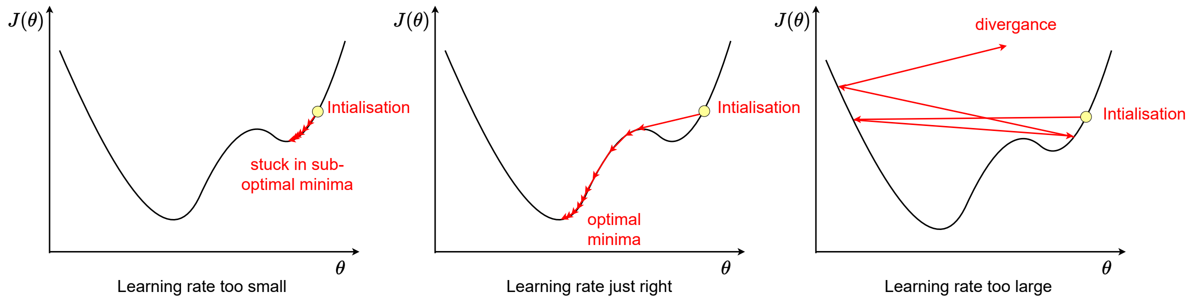


Figure 3.14: Single parameter model updates (red arrows) for various learning rates and subsequent observed effects during training. Note we omit the loss functions dependence on $\boldsymbol{x}$ for simplicity.

stuck in a sub-optimal solution, and if it is too large, training will diverge. Practically, loss surfaces have high dimensionality and are problem specific, and as such the general rule of thumb is to simply try a few learning rate values until adequate performance is achieved. A manual method such as this is not viable for large-scale experimentation which will be conducted in later chapters.

One such method that attempts to semi-automate learning rate selection utilises a learning rate sweep performed at the start of training. The learning rate is linearly increased each step from zero (whilst updating model parameters after each step) until the loss diverges or the maximum number of steps is reached. The learning rate that corresponds to the steepest gradient (implying parameters are being updated most efficiently) from the resulting loss versus learning rate function is chosen [55]. We visualise this technique with a stylised example shown in Figure 3.15, and observe that the learning rate corresponding to the steepest decrease in loss (not the minimum value) is selected. Importantly, the model is reinitialised after the sweep and training is resumed with the newly determined optimal learning rate.

3.7.4. Over-fitting

Over-fitting is a phenomenon in machine learning whereby spurious correlations that exist in training data are relied on to achieve a more optimal solution compared to that which would have been achieved if only the true signal was learnt. When this happens, a model is deemed to have *over-fit* to the training data. Such a model can subsequently fail to generalise to unseen data in which such spurious correlations do not exist, which can become problematic.

We provide a simple example to illustrate this premise in its most basic form. For a given data set, two predictors *hair colour* $\in$ [red, brown, black, blond] and *number of cars* $\in \mathbb{W}$, are used predict a person's net-worth. Due to the small sample size, there exist a few spurious relationships in the data. Namely, people with red hair always have fewer than two cars but a high net worth and; those with black hair who all have two cars or more

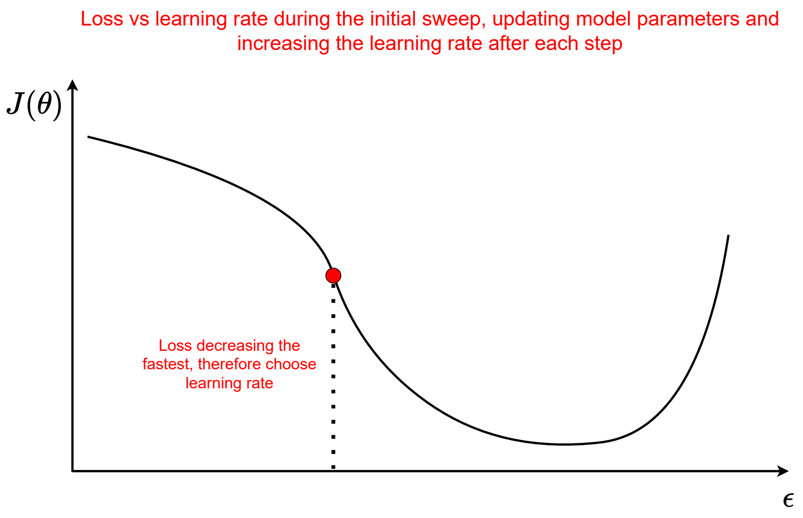


Figure 3.15: Stylised learning rate sweep, illustrating how the optimal learning rate is selected. Note after each step model parameters are updated and the learning rate is increased.

always have a low net worth. The informed reader will understand that a person's hair colour is uncorrelated with their net worth, whilst the number of cars they own is a somewhat correlated yet noisy predictor. However, due to the noise in the true signal that we intend to learn, the model might rely on the spurious hair colour phenomena in the training data to reduce the error in the function it fits, which is of course its only objective. Come test time, this will allow hair colour to incorrectly contribute to the net worth predictions of unseen samples, degrading the reliability of those predictions. In this over-simplified example, it would be simple to negate this effect by simply removing the troublesome hair colour predictor entirely. In practice, this is not simple and is made even more complex when such spurious correlations only become evident after many non-linear transforms.

Whilst this phenomenon is not unique to deep learning, it is of particular concern when working with such models due to the relative ease of it taking place. Particularly, this is due to their typically large number of parameters (which often exceed the number of training examples) which, in conjunction with the characteristics of the activation functions with the architecture, enable the ability to learn highly non-linear relationships which can easily become spurious. Moreover, these relationships are often impossible to observe when the data is in its base state (unlike the previously described example). Techniques such as restricting the number of features and decreasing model size are commonly used to address this issue, in conjunction with additions to model architectures which include dropout layers, normalisation layers, and weight regularisation (all used in this work) which are described in more detail subsequently.

Dropout

Dropout is a regularisation technique whereby each element of a specific layer's output (and subsequent input to the dropout layer) has a certain probability of being "dropped" (being multiplied by zero), the specific probability of which is a user-defined hyper-parameter. By randomly omitting subsets of layer outputs, the neurons that make up this layer cannot rely on specific combinations of one another to form complex relationships that map specific training samples to their respective targets, but instead, each neuron is forced to learn a general feature mapping that is in some way helpful to produce the target during training [56].

Batch and layer normalisation

Batch normalisation is a technique used to either normalise (across a mini-batch) the inputs to a network, or the outputs of intermediary hidden layers. The standardisation across mini-batches helps stabilise training by reducing internal covariate shift - the change in the distribution of network activations during training which can result in activation saturation - in addition to reducing the ability of models to fit to noise present in training samples [57]. The normalisation process is described in Equation 3.19 where $\boldsymbol{x} \in \mathbb{R}^B$ is the vector containing the activations for each sample in a mini-batch of size B from a particular neuron and $\bar{x}$ and $\sigma_{\boldsymbol{x}}$ are it's mean and standard deviation along the batch dimension. Both γ and β are optional learnable parameters.

$$\text{BatchNorm}(\boldsymbol{x}) = \frac{\boldsymbol{x} - \bar{x}}{\sigma_{\boldsymbol{x}}} \cdot \gamma + \beta \tag{3.19}$$

In contrast, layer normalisation performs this normalisation not on the batch dimension, but rather along each sample's feature dimension. This is accomplished using the exact same equation and process with the exception that $\boldsymbol{x} \in \mathbb{R}^d$ is the vector of d activations in a specific layer for a specific training sample.

Weight regularisation

Weight regularisation, also known as weight decay, is a regularisation term added to a model's loss which is the L2 norm of the model's weights. This results in the model minimising both the objective loss function and this L2 norm. By penalising large weights, the model's freedom to over-fit to the training samples is reduced in addition to helping to prevent exploding gradients.

3.8. Acoustic features

In most audio recordings unwanted attributes are present which can include characteristics such as speaker stressing, background noise and amplitude variations [58]. To separate desired information from some of these unwanted attributes and to uncover information not initially evident in the audio itself (such as frequency content), acoustic feature vectors are typically used to represent frames of audio. Generally, the continuous audio signal is windowed, resulting in a segment of audio referred to as a frame, upon which feature extraction is performed. This window then moves forward by some amount of time or samples, yielding a new frame upon which feature extraction is performed again. The amount the window is shifted is known as the hop length and varies depending on the type of feature extraction being performed. In previous work, two acoustic feature vectors have been used to aid in the successful classification of TB coughs, MFCCs and LFB energies [2, 3]. Both are described subsequently. In addition, we also describe mel-spectrograms, a prevalent acoustic feature representation used in many other audio domains, especially for neural networks.

3.8.1. Linear filterbank energies

Audio signals are often best described in terms of the change in their frequency information over time. Typically this is represented in the form of a STFT, whereby a window is slid over the acoustic signal; each time computing the Fourier transform of that frame of audio, yielding a snap-shot of frequency information. However, depending on the frame size, this can result in a very high dimensional representation, which is generally not ideal when training deep neural networks, specifically in low-resource applications. For this reason, several acoustic feature extraction techniques exist that both aim at reducing the dimensionality of this feature space whilst retaining relevant information to the task at hand.

One of the most simple of these acoustic feature representations is a linearly scaled filter bank (LFB). First, the STFT of an audio signal is computed with the desired frame size and hop length. Next, the log energy spectrum of each frame is multiplied by a set of N_{LFB} overlapping triangular filters, as shown in Figure 3.16, normalised such that each filter has unit area. Finally, the weighted sum of the resulting scaled log energy spectral coefficients is determined for each filter, resulting in a vector of N_{LFB} log spectral energies.

3.8.2. Mel-spectrograms

Similarly to LFB energies, the mel-spectrogram is also the result of applying a set of N_{mel} triangular filters to a STFT. However, these are not linearly spaced, but rather spaced according to the mel scale f_{mel}. The mel scale follows a logarithmic-like function which

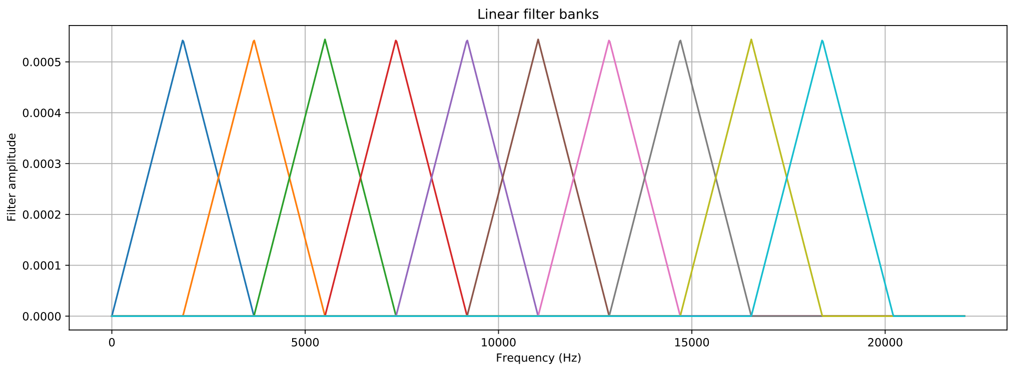


Figure 3.16: An example of a linearly scaled filter bank. Each colour represents an individual filter.

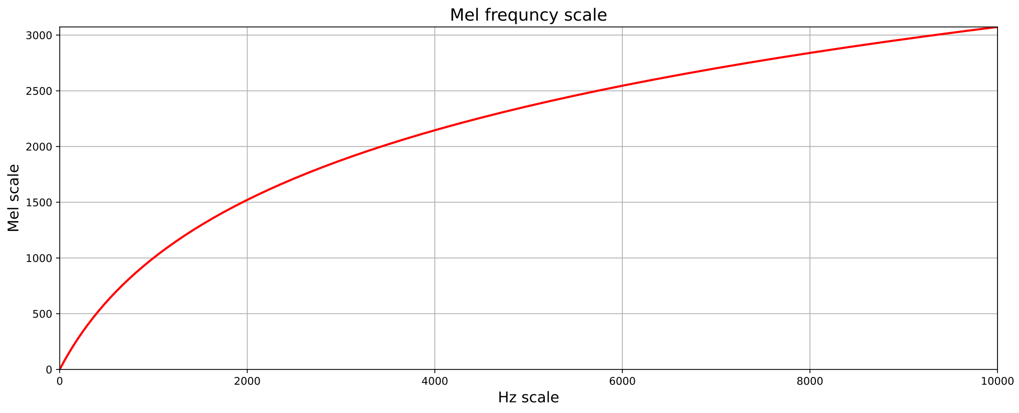


Figure 3.17: Depiction of the mel scale.

maps pitches perceived to be equal distances from one another by human listeners. The scale is described by Equation 3.20 which is further depicted in Figure 3.17. The mel scale is used in a variety of speech applications, due to its ability to mimic the spectral resolution of the human ear.

$$f_{mel} = 1127 \cdot \ln\left(1 + \frac{f}{700}\right) \tag{3.20}$$

Mel-scaled triangular filters are visualised in Figure 3.18. Note that the peak of each filter is proportional to its width, such that the area in each is the same. Following the same process described for LFB energies, the log STFT spectrum is multiplied by the filter bank, and the energies for each respective filter are summed, resulting in a N_{mel}-dimensional feature vector of mel scaled log spectral energies (referred to as a mel-spectrogram).

3.8.3. Mel frequency cepstral coefficients

MFCCs capture spectral information, whilst maintaining a low dimensionality by only capturing the perceptually important characteristics of speech [59]. This is accomplished

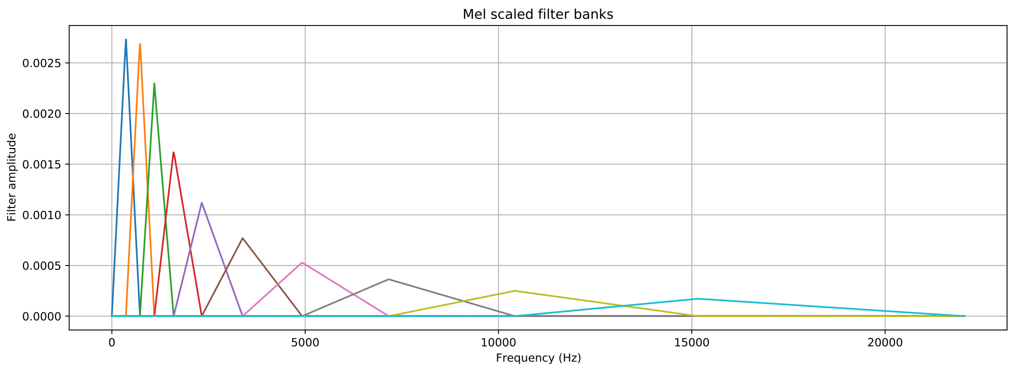


Figure 3.18: Mel scaled filter banks. Each colour represents an individual filter.

by first computing the STFT of the audio signal. A mel-scaled filter bank is then applied to the spectra, in the exact way described for mel-spectrograms as shown in Figure 3.18. The logarithm of these filter bank outputs ($\boldsymbol{S}_k$) is then transformed back to the time domain using the discrete cosine transform (DCT) which outputs n cepstral coefficients shown in Equation 3.21, where K is the number of filter banks [60]:

$$C[m] = \sum_{k=0}^{K-1} \log_{10}(\boldsymbol{S}_k) \cos\left(\frac{\pi m(k-0.5)}{K}\right), \quad m = 0, .., n \tag{3.21}$$

The elements of the resulting $n+1$-dimensional acoustic feature vector is named the mel frequency cepstral coefficients. Empirically, $n = 13$ has been found to be optimal for speech processing since it captures the aspects of the audio signal that are discernable by the human ear and are important for the recognition of speech. However, this value can be increased to compute higher-resolution MFCCs. To capture the change of these features over time, the first and second derivatives, often referred to as velocity and acceleration, of these cepstra are appended to the feature vector. These derivatives are approximated using N preceding and succeeding frames as shown in Equation 3.22. For 13 MFCCs, this results in a 39-dimensional acoustic feature vector for each frame [61].

$$d_t = \frac{\sum\limits_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum\limits_{n=1}^{N} n^2} \tag{3.22}$$

In Equation 3.22, d_t is the velocity coefficient for the cepstral coefficient c_t of frame t computed in terms of the neighbouring coefficients from frame $t-N$ to $t+N$. The resulting velocities are used to estimate the acceleration using the same equation.

3.8.4. Feature normalisation

It is well known in machine learning literature that features normalisation can help stabilise training. Particularly in deep learning, the gradients of non-linear activation functions can saturate for large inputs, making it hard to converge to local minima through gradient-based learning methods. For this reason, all acoustic features are normalised to lie in the range $[-1, 1]$ using global max/min scaling shown in Equation 3.23. It should be noted that there are a vast number of feature normalisation techniques provided in the literature (for example, statistics pooling). However, the investigation of these was not the focus of this work and as such we opt for the simplistic aforementioned approach.

$$\boldsymbol{X}^{(n)}_{\text{norm}} = \frac{\boldsymbol{X}^{(n)} - \max(\forall \boldsymbol{X})}{\max(\forall \boldsymbol{X}) - \min(\forall \boldsymbol{X})} \tag{3.23}$$

This equation shows how the feature vectors are scaled by the largest and smallest values present in the training dataset. This is shown for a single acoustic feature vector $\boldsymbol{X}^{(n)}$ in the set of acoustic feature vectors $\boldsymbol{X}$ in Equation 3.23.

3.9. Summary

In this chapter, we introduced the fundamental deep learning architectural building blocks used in this work, namely, fully-connected neural networks (FNNs), convolutional neural networks (CNNs), ling short-term memory networks (LSTMs), attention, and transformers. Next, we described the typical training procedure to fit such architectures. Lastly, we detailed three acoustic feature vectors explored in this work, which include linear-filter bank energies, mel-spectrograms, and mel-frequency cepstral coefficients. In the next chapter, we provide an overview of the datasets used for experimentation in this work.

Chapter 4

Datasets

In this chapter, we first detail two datasets compiled in previous TB cough classification studies, namely the Brooklyn and Wallacedene datasets [2, 3]. The unique characteristics of each are given, and the data collection scheme employed by each study is described. Next, we detail and motivate the combination of these datasets, which will be used for experimentation in this work. Finally, we describe a dataset compiled for use during pre-training, titled "Cough or Not", in which audio samples are labelled as either coughs or some other environmental noise.

4.1. Brooklyn dataset

The Brooklyn dataset [2] comprises coughs originating from TB-positive patients (confirmed before recordings) in addition to healthy controls, who were close contacts of the patients. Data collection was performed in a controlled environment in the form of a specially-designed noise-isolated facility. Recordings were made with a high-quality studio microphone with a sample rate of 48kHz, placed at a distance of between 15cm – 40cm from the seated coughers.

In total, there were 38 participants of which 17 were TB positive and 21 were healthy controls (henceforth referred to as TB and $\overline{\text{TB}}$). The details of the collected data are presented in Table 4.1. In total, 746 coughs were collected resulting in 377 seconds of audio. Despite fewer TB than $\overline{\text{TB}}$ participants, we note more than double the number of TB coughs. This is however expected, as the healthy controls did not suffer from any respiratory diseases and therefore found it harder to cough. The Brooklyn dataset also includes clinical measurements such as BMI, heart rate and temperature, but was not used in this work and hence are not further detailed.

4.2. Wallacedene dataset

The Wallacedene dataset [3] consists of coughs collected from patients that reported to a primary health clinic, all of whom were ill and suspected of having TB at the time of recording. Recordings were made in an external booth typically used for sputum analysis

Table 4.1: Brooklyn dataset description.

	TB	$\overline{\text{TB}}$	Total
Patients	17	21	38
Total coughs	501	247	748
Mean coughs per patient	29.47	11.76	19.68
Std dev coughs per patient	22.56	10.89	19.25
Mean cough length (s)	0.55	0.40	0.50
Std dev cough length (s)	0.22	0.16	0.22

Table 4.2: Wallacedene dataset description.

	TB	$\overline{\text{TB}}$	Total
Patients	11	25	36
Total coughs	219	637	856
Mean coughs per patient	19.91	25.48	23.78
Std coughs per patient	7.48	11.44	10.71
Mean cough length (s)	0.83	0.68	0.72
Std dev cough length (s)	0.32	0.35	0.35

adjacent to a busy road with minimal external noise isolation. Since the goal of the dataset collection scheme was to mimic the *real-world* scenario in which a field TB screening tool would be deployed, no effort was made concerning noise isolation or reduction. With this in mind, the recordings contain significant background noise, ranging from chatter, vehicles, pets, and even music. However, when performing an initial exploratory investigation into the composition of the background noises present in the dataset, we found that some recordings contained severe distortions and disturbances. These were unrelated to the background noise. The process by which this was dealt with, together with a summary of the *clean* dataset, will be detailed in the next section.

Recordings were made with a high-quality field recorder, at a sampling rate of 48kHz. Patients were placed $10\text{cm} - 15\text{cm}$ from the microphone and were asked to produce at least two bursts of voluntary coughing, although typically the number of coughing episodes was significantly higher due to the irritation in the patients' lungs associated with their ailment. In total, recordings from 49 patients were collected of whom 16 were later confirmed via sputum analysis as being TB positive whilst 33 were determined to be TB negative.

4.2.1. Data cleaning

As previously mentioned, some recordings contained severe distortion caused for example by shaking of the microphone stand, amplitude saturation, or 'popping'. Whilst the original dataset consisted of 49 patients, 13 of these recordings were identified as containing such distortions. This was determined by manually listening to each recording, and categorising

Table 4.3: Combined dataset description.

	TB	$\overline{\text{TB}}$	Total
Patients	28	46	74
Total coughs	720	884	1604
Mean coughs per patient	25.71	19.22	21.68
Std coughs per patient	18.78	13.11	15.82
Mean cough length (s)	0.64	0.60	0.62
Std dev cough length (s)	0.29	0.34	0.32

them into either: ‘no disturbance’, ‘some disturbance’, or ‘significant disturbance’. Since these disturbances could mask important frequency information, all recordings labelled as ‘significant disturbance’ were removed from the dataset. A list including the notes and noise labels for each recording can be found in Appendix D. After cleaning, 11 TB-positive and 25 TB-negative patients remained. It should be emphasised that recordings were only removed based on audio signal distortions and not background noise (music, chatter, vehicles etc.). We detail the cleaned Wallacedene dataset in Table 4.2. In contrast to the Brooklyn dataset, we observe that the cough rate for $\overline{\text{TB}}$ patients is no longer lower than for TB patients. Considering that both groups of patients suffering from lung ailments and therefore are predisposed to coughing, these two rates should in fact be similar. Upon further investigation, it became clear that these cough rates are in fact much closer (25 and 26 respectively) when including the recordings with severe distortion. This suggests that the noisy TB recordings appear to be associated with prolonged (and perhaps more violent) coughing episodes. Further investigation is necessary to verify this, however.

4.3. Combined dataset

Previous work in TB cough classification has relied on relatively small datasets gathered in a single recording environment from a small number of patients. This is problematic when training deep architectures due to their tendency to over-fit, especially in a clinical setting, to confounding factors [62]. Consequently relying on recordings from a single environment restricts data diversity and consequently the final model’s ability to generalise. Additionally, by combining datasets we can begin to quantify performance when tasked with detecting TB coughs from multiple recording domains and provide a small insight into expected performance in a real-world application, whereby the recording conditions will certainly change. In an attempt to address this, we combine the Brooklyn and Wallacedene datasets used previously in [2] and in [3]. This is in an effort to yield a more environmentally diverse dataset. This combined dataset is summarised in Table 4.3. In total, the dataset consists of 74 patients, 28 of whom have TB, while the remaining 46 belong to the $\overline{\text{TB}}$ set. This doubles the size of datasets used in previous work.

4.4. "Cough or Not" dataset

Table 4.4: "Cough or Not" dataset description for each audio source.

Dataset	Samples	Length (hrs)
TASK (other)	13208	2.55
AudioSet (other)	1570	4.37
TASK (coughs)	2816	0.72
Free sound (coughs)	249	0.52
AudioSet (coughs)	410	1.13
COUGHVID (coughs)	1565	3.65
Total not cough	14778	6.92
Total cough	5040	6.00

In addition to the dataset described in the previous section, we compiled a dataset for a pre-training task, with the aim of improving model performance on the primary TB classification task. We choose cough detection as our pre-training task as also done in [34] and use a similar set of data (without the inclusion of LibriSpeech). Since the goal of the pre-training task is to allow for the transfer of some learnt information to the primary TB classification task we also include voiced sounds that may be acoustically similar to cough in the "other" category such as laughing, sneezing and throat clearing. This is in an effort to encourage networks to learn more complex relationships that can help to distinguish between coughing and these other similar sounds during pre-training. We detail each audio source below and provide a summary of the combined "Cough or Not" dataset in Table 4.4. In total, the combined dataset contains 776 minutes of audio of which 361 minutes contain coughing and the rest consists of other environmental sounds.

TASK: The TASK dataset was collected in a TB ward in Cape Town, South Africa that consists of both coughing and environmental noises (doors opening, beds shuffling, chatter and other ward background noise). Recordings were made using smartphones with external microphones attached to the patients' beds [63].

COUGHVID: COUGHVID is a corpus of crowd-sourced COVID-19 coughs collected throughout the pandemic whereby participants submitted cough recordings from across the globe via an online platform [38]. In total, $25,000$ recordings were gathered, of which we use a randomly selected small subset simply because it was easily available from prior work.

AudioSet: Google's AudioSet is a large audio dataset with over 2 million recordings hand-annotated into 632 audio event classes. the recordings were gathered from YouTube videos covering a broad range of recording environments [43]. We use a subset of this dataset as a source of both "other" and "cough" samples. Specifically, we use sneeze, laugh and throat-clearing classes as "other" samples in our dataset, in addition to using the cough class recordings.

Freesound: Freesound is a large, collaborative database containing an assortment of audio recordings, snippets, effects and samples released under the creative commons license [36]. We use recordings of coughs uploaded to this platform to diversify our dataset.

Due to the variation in sample rate across different audio sources (ranging from 16 to 48kHz), all samples are down-sampled to 16KHz. Importantly, all data used subsequently to train TB classifiers which used the pre-trained neural networks were down-sampled to the same frequency.

4.5. Summary

In this chapter, we have presented an overview of the datasets used in previous research into TB cough classification, namely the Brooklyn and Wallacedene datasets. We then detailed the amalgamation of these two sources of data to produce the dataset used for experimentation in this study. This combined dataset includes 76 patients, double the number used in previous studies. The dataset is also more representative of diverse recording environments. We also highlighted the presence of some severe distortions in some of the recordings in the Wallacedene dataset and detailed how these were removed from the dataset used in this work. Lastly, a large dataset constructed for a pre-training task was described (classifying audio as either containing cough or not) which totalled 13 hours of audio recordings from a multitude of sources.

Chapter 5

Pre-trained Networks

In Chapter 2 we briefly introduced prominent pre-trained networks. We now describe these networks in detail for the sake of brevity in later chapters and motivate their suitability for TB cough classification. Detailed descriptions are provided for ResNet [4], Audio Spectrogram Transformer [5], and wav2vec2.0 [64].

5.1. ResNet

CNN architectures have become the dominant choice for vision-based tasks [65–67]. Generally, such networks perform better as the number of layers (and therefore model size) increases. This makes it hard to train state-of-the-art vision models from scratch without huge computational resources. As such, it is common to begin with a network that has been pre-trained on large amounts of data, and fine-tune it to the target domain. One architecture that is widely used in this way is ResNet [4], a CNN which achieved state-of-the-art performance on ImageNet, a dataset that spans 1,000 object classes and more than one million training images [68]. All proposed ResNet architectures from ResNet-34 onwards improved upon the previous best methods, with ResNet-152 achieving a top-1 error rate of 19.38% - a 11.87% relative reduction. Whilst ResNet is no longer the top performer, it remains widely used as a backbone in many visual tasks due to its relative simplicity and availability. This includes tasks such as object detection, semantic segmentation, pose estimation and other image classification tasks [69].

Particularly, ResNet is a convolution-based framework which eases the training of large CNN-based image networks by incorporating skip connections between intermediary convolutional blocks. In addition to contributing to faster optimisation, the additional residual connections allow the depth of such networks to increase to previously unrealistic values whilst not suffering from vanishing gradients. An example of a 34-layer ResNet (ResNet-34) is provided in Figure 5.1. The layers between skip-connections are referred to as *convolutional building blocks.* Every ResNet architecture starts with an initial convolutional layer with 7×7 filters, stride 2 and 64 channels followed by a 3×3 max pooling layer with stride 2. These together compress the $3 \times 244 \times 244$ sized RGB input image. These are then followed by a series of building block layers that intermittently

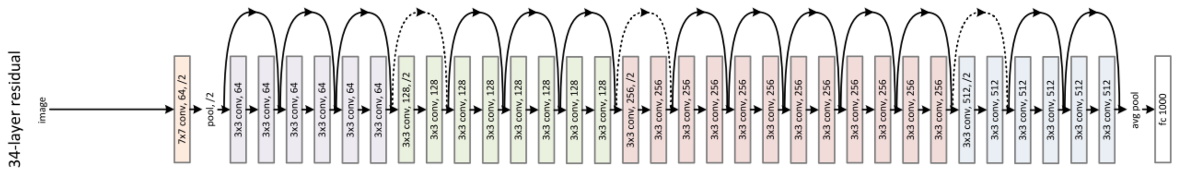


Figure 5.1: An illustration of the 34-layer ResNet architecture (ResNet-34). Dotted shortcuts indicate an increase in channel dimension. Reproduced from [4].

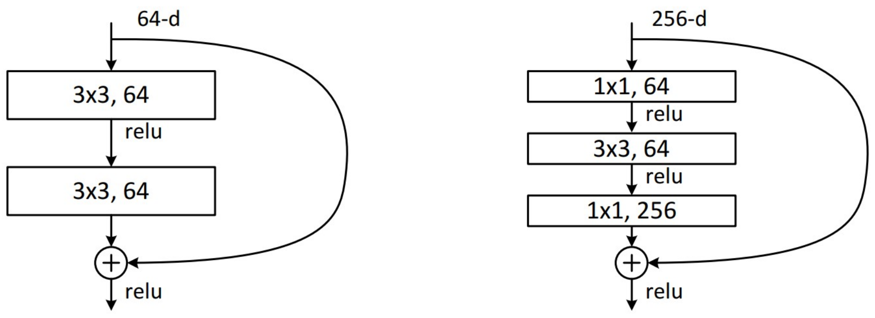


Figure 5.2: A residual building block used in ResNet-18/34 (left) and the bottleneck residual building block used in ResNet-50/101/152 (right). Reproduced from [4].

increase the channel number. In the ResNet-18 and ResNet-34 architectures, each building block is made of two convolutional layers with 3×3 filters, where the number of channels is doubled every 2 blocks for ResNet-18, and after the 3rd, 7th and 13th building block for the ResNet-34 architecture. ResNet-50 follows the same pattern of channel dimension increases as ResNet-34, however, it uses a *bottleneck* design for each building block to decrease the computational cost and parameter count of these deeper networks. Each building block consists of three convolutional layers: a layer with 1×1 kernels to reduce the number of channels by a factor of four, a layer with kernel size 3×3, and a final 1×1 layer that up-samples the number of channels back to the original dimension. A similar process is followed for ResNet-101 and ResNet-152. Both building blocks are visualised in Figure 5.2, showing the input and output dimensionality (number of channels).

5.1.1. Motivation

As presented in Chapter 2, success has been achieved in the literature when using pre-trained ResNet architectures for cough classification. As such, we investigate its application to TB cough classification. Typically in the literature, ResNet-50 was used, which has over 23 million trainable parameters. Due to the small nature of the dataset and the tendency for over-parameterised models to over-fit, we also investigate the application of the smaller ResNet-18 architecture (11 million parameters).

5.2. Audio Spectrogram Transformer

Vision-based models have remained popular in audio classification tasks due to their ability to locate acoustic signatures in a robust manner. Recent advancements in transformer-based vision models have resulted in their dominance in state-of-the-art computer vision research [42, 70]. Consequently, the Audio Spectrogram Transformer (AST), which is built upon a pre-trained vision transformer model, has achieved state-of-the-art results on audio classification benchmarks [5].

AST's architecture is shown in Figure 5.3, where it follows the same structure as the Vision Transformer (ViT) architecture. In ViT, patches of an image are flattened and linearly projected onto an embedding, positionally encoded, and fed through a large transformer which extracts the non-linear latent relations between patches and embeds them in a classification token which can be used for downstream tasks. In fact, AST fine-tunes a DeiT-base model [71]; a distilled vision transformer pre-trained on ImageNet [42]. A fundamental underlying problem with the structure of the ViT architecture is that it lacks the inductive biases CNNs possess, and therefore requires much larger amounts of data (and compute power) to generalise to image tasks. DeiT builds upon the ViT architecture by successfully using substantially smaller architectures that are less data-hungry by employing a knowledge distillation *teacher-learner* paradigm during a pre-training step. The DeiT model (learner) is initialised to mimic the predictions of an already trained CNN (teacher). This allows the DeiT model to quickly learn the desirable properties important for computer vision which are inherent to CNNs (such as being invariant to translation and location) without needing copious amounts of data. Whilst three DeiT variants were proposed, AST utilises DieT-base (the largest of the three) which comprises a 12-layer transformer encoder with 12 attention heads.

AST is trained on AudioSet, which contains 5.5k hours of sounds from 527 classes extracted from YouTube videos [43]. First, mel-spectrograms are divided into 16×16 patches (with an overlap of 6 found to be optimal). Each patch is flattened $\boldsymbol{x}_p \in \mathbb{R}^{256}$ and projected with a point-wise linear layer to a latent patch embedding $\boldsymbol{z}_p \in \mathbb{R}^{768}$. Learnable positional embeddings are then added to each patch embedding. Since the original DeiT architecture was trained with image resolutions that differ from the mel-spectrograms presented to AST, the learnt positional tokens $\boldsymbol{E} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_P\}$, where $\boldsymbol{e}_p \in \mathbb{R}^{768}$ and P is the number of patches, cannot simply be reused. The authors propose a cut and bilinear interpolation approach to transfer the learnt embeddings to different input dimensions. For example, in the case of the original DeiT-base model, each 384×384 image with patch sizes of 16×16 results in 24×24 individual patches and therefore the same number of positional embeddings. However, in the case of a 128-dimensional mel-spectrogram with 1024 frames, the resulting number of patches would be 12×100 (with a patch overlap of 6). Therefore, to maintain the information that the pre-trained positional embeddings have already learnt, the first dimension of the original 24×24 embeddings (arranged into a grid) is reduced to 12, and the second is interpolated to 100. This process is visualised in Figure 5.4.

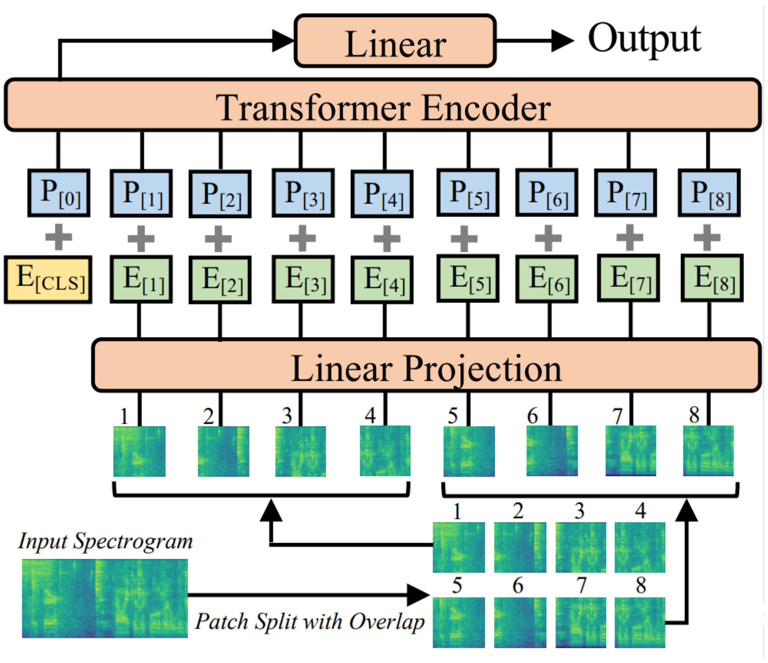


Figure 5.3: High-level diagram of AST. A 2D spectrogram is split up and linearly projected into patch embeddings where an additional classification token is prepended. The resulting sequence is combined with learnable positional embeddings and fed into a transformer encoder. The output corresponding to the classification token is then fed into a linear layer for classification. Reproduced from [5].

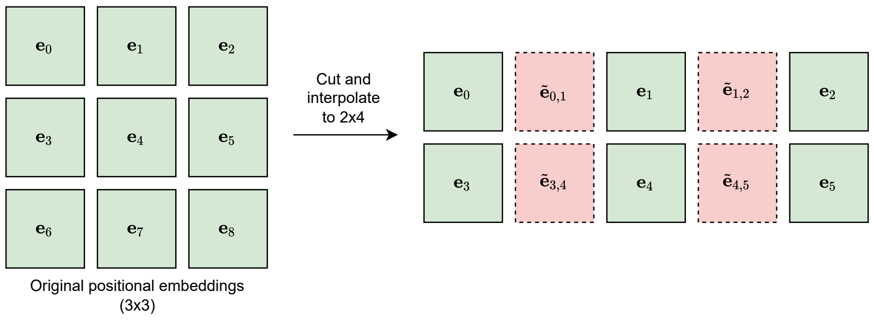

e0 e1 e2
e3 e4 e5
e6 e7 e8
Original positional embeddings (3x3)
Cut and interpolate to 2x4
e0 ẽ0,1 e1 ẽ1,2 e2
e3 ẽ3,4 e4 ẽ4,5 e5

Figure 5.4: Visualisation of reducing the dimension of a matrix along one axis, and interpolating it along another. Red squares indicate interpolated samples where $\tilde{\boldsymbol{e}}_{i,j}$ is the mean between $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$.

5.2.1. Motivation

Large transformers such as AST benefit from having been trained on exceptionally large datasets. Consequently, training such a large transformer from scratch would not be feasible with the size of the dataset used for experimentation in this work. With the success of fine-tuning pre-trained out-of-domain architectures on seemingly unrelated tasks, for example, the previously highlighted widespread use of ResNet in COVID-19 cough classification, it was hoped that similar success could be achieved when fine-tuning AST (which has already learned general audio representations). As such, we investigate its use for TB cough classification during experimentation.

5.3. wav2vec2.0

Large transformer-based language models have proven successful in capturing complex contextualised representations resulting in state-of-the-art performance on a multitude of ASR tasks [72]. A highly successful architecture is wav2vec2.0 [64], which starts by extracting a sequence of acoustic representations $\boldsymbol{Z} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_t\}$, where $\boldsymbol{z} \in \mathbb{R}^d$, directly from an audio waveform by means of a 1D convolutional feature extractor. This feature extractor consists of seven layers, each with 512 channels, with the kernel width and stride of 10,3,3,3,3,2,2 and 5,2,2,2,2,2,2 respectively, that reduce the size of the normalised input waveform (unit mean and variance) by a factor of 320. This is then followed by a normalisation and projection layer (a point-wise linear layer), which projects the 512-dimensional embeddings to be d-dimensional embeddings, where d is either 768 or 1024 depending on the size of the network. $\boldsymbol{Z}$ is then fed into a series of transformer encoder blocks, which extract contextualised representations $\boldsymbol{C} = \{\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_t\}$, where $\boldsymbol{c} \in \mathbb{R}^d$, as shown in Figure 5.5. To circumvent the large labelled dataset that would be needed to train the model from scratch, a self-supervised learning technique is employed that allows the model to learn generalised speech representations using a contrastive loss. During this self-supervised training, feature representations $\boldsymbol{Z}$ are discretized into the set of vectors $\boldsymbol{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_t\}$ where $\boldsymbol{q} \in \mathbb{R}^d$ using a complex product quantization scheme to enforce the learning of speech-like acoustic units. The details of this procedure are not necessary for a high-level understanding of the wav2vec2.0 framework. Next, a proportion of these quantized representations $\boldsymbol{Q}$ are masked (replaced with a learnable masking token) before being fed into a transformer encoder. The network is trained to minimise the cosine distance between the context representations $\boldsymbol{C}$ generated for these masked representations at the output of the transformer and the corresponding quantized speech representations $\boldsymbol{q}$. Additionally, the cosine distance is maximised between the generated context representations and a set of distractors, which are true quantized speech representations uniformly sampled from other masked time steps in a given utterance. This is described by the following equation, where $\boldsymbol{c}_j$ is the extracted context representation for some masked speech representation, $\boldsymbol{q}_j$ is its quantized speech representation and $\tilde{\boldsymbol{q}}$ is a distractor form a set of K distractors.

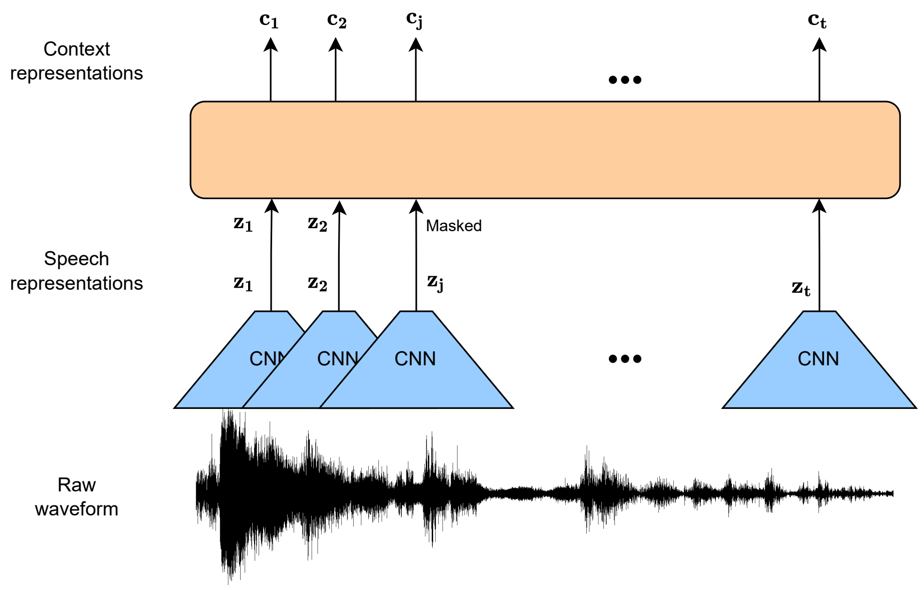


Figure 5.5: Illustration of the wav2wec2.0 architecture. A raw waveform is fed into a convolutional speech representation extractor, the output of which is used by a transformer-based context network.

$$\mathcal{L} = -\log \frac{\exp(sim(\boldsymbol{c}_j, \boldsymbol{q}_j)/K)}{\sum_{\forall \tilde{\boldsymbol{q}}} \exp(sim(\boldsymbol{c}_j, \tilde{\boldsymbol{q}})/K)} \tag{5.1}$$

Two variants were released, namely “base” (12 transformer blocks) and “large” (24 transformer blocks), trained on 960 and 53,2k hours of unlabelled audio derived from the LibriSpeech and LibriVox corpora respectively. Finally, these architectures were fine-tuned with (comparably less) labelled data using connectionist temporal classification (CTC) loss to form acoustic models in ASR tasks where they achieved state-of-the-art results. In addition to their great success in ASR, the latent acoustic representations learnt during the self-supervised pre-training step have been shown to transfer well to a variety of acoustic tasks including phoneme recognition, keyword spotting, speaker identification, speaker verification, speaker diarization, intent classification, semantics, and emotion recognition [73].

5.3.1. Motivation

Despite cough being characteristically different from speech, it has been shown in previous studies that the important frequency information being learnt lies within the same bands as speech. Moreover, motivated by the success of transfer learning in other cough classification studies, a fine-tuned wav2vec2.0 may be able to capture good acoustic representations useful for TB cough classification. To investigate this hypothesis, we use the base variant of wav2vec2.0 in experimentation.

5.4. Summary

In this chapter, we detailed three large pre-trained networks used in this work, namely ResNet, AST, and wav2vec2.0. ResNet is a large convolutional-based architecture that has been used for a variety of downstream image-based tasks and is pre-trained on ImageNet. AST, which applies transfer learning to an already pre-trained vision transformer on ImageNet, performs convolution-free audio classification. Lastly, wav2vec2.0 is another transformer-based acoustic model. Rather than using image patches (like AST), wav2vec2.0 extracts acoustic representations directly from the audio waveform and can therefore be used for a host of downstream acoustic tasks, most notably ASR. In the next chapter, we detail the experimental setup, including architecture design, for the primary experimentation conducted in this study, that is, the application of deep learning-based methods to TB cough classification.

Chapter 6

Experimental setup

In this chapter, we detail the application of deep learning -based models to TB classification, which is the main focus of this work. First, utilising the deep learning architectural building blocks and pre-trained networks described in previous chapters, investigated architectures and variations of which (structure, depth, complexity, and size) are detailed. In addition to training randomly initialised classifiers, we investigate the effect of pre-training models on an auxiliary task before fine-tuning them for TB classification. We detail this pre-training scheme, a cough detection task, whereby models are exposed to significantly more data. Next, investigated hyper-parameters for the three previously described acoustic feature vectors (LFB energies, mel-spectrograms, and MFCCs) are detailed, and the investigated data augmentation techniques used are described. This is followed by a thorough description of the experimental setup used, which includes dataset splits and the use of nested cross-validation, performance metrics (sensitivity, specificity, equal error rate, and area under the receiver operating characteristic curve), and the general training procedure.

6.1. Architectures

Previous work has shown that machine learning algorithms can learn to distinguish between coughs originating from patients that suffer from TB and those that do not. We conduct a rigorous investigation into the application of deep learning to this domain, comparing a variety of architectures to the current best classifier that has been presented in previous work (LR). These architectures include RNNs, CNNs, transformers, the use of attention, and various larger models pre-trained on seemingly unrelated tasks which we fine-tune for TB cough classification. We subsequently describe the exact classifier architectures used for experimentation in this work. Albeit each pre-trained network or architectural building block has already been explained in previous chapters in-depth (Chapter 5 and Chapter 3), we provide a brief motivation and description of each when used. Where this detail is insufficient, we refer the reader to the relevant chapters. This description also includes approaches to architecture optimisation. We detail the general method used to train these deep learning architectures in later sections. Unless otherwise stated, sequences of acoustic feature vectors are always zero-padded to the longest sequence in a given mini-batch.

6.1.1. Linear Regression

Previous work in TB cough classification has focused on simple linear models since it was observed that more complex neural networks resulted in degraded performance [3]. In both [2] and [3], logistic regression (LR) outperformed all other classifiers considered. As such, we use it as a baseline with which our architectures will be compared. LR is a simple approach that linearly models a probability $f(\boldsymbol{x})$ given a set of d predictors $\boldsymbol{x} \in \mathbb{R}^{d_x}$ using learnable coefficients $\boldsymbol{\theta}$.

$$f(\boldsymbol{x}) = \frac{1}{1 + e^{\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}}} \tag{6.1}$$

Equation 6.1 highlights an important limitation of LR: each predictor $\mathbf{x}$ is a feature vector computed from a segment of audio - the frame-wise average of a sequence of acoustic feature vectors. In doing so, any temporal information is lost. In this work, we use this same procedure to ensure a fixed dimensional input feature vector, that is, for each segment of audio, we compute the temporal mean of its corresponding sequence of acoustic feature vectors. Moreover, whilst previous studies experimented with dividing a single cough audio sample into multiple segments (refer to Chapter 2 for more detail), one segment was always optimal, and as such we fix this hyper-parameter in our experimentation.

Hyper-parameters

There are no architectural hyper-parameters for LR, but it is common to introduce regularisation to improve generalisation. Typically, an additional loss term is introduced that penalises the magnitudes of learnt parameters which helps prevent over-fitting. This is performed by either minimising the square magnitude or the absolute magnitude of the coefficient vector (known as L2 and L1 regularisation respectively). The latter has the benefit of allowing coefficients to become zero, allowing the model to perform its own feature selection which is very useful in low-resource applications. To remain consistent with both [2] and [3], we use a linear combination of L2 and L1 (known as an elastic net), performing a grid-search between the following parameters: The overall regularisation strength (weighting) λ and relative weighting between L1 and L2 regularisation β_r. A β_r value of 0 corresponds to only using L2 regularisation, and a value of 1 corresponds to only using L1 regularisation.

Optimisation

We use the saga solver in scikit-learn for logistic regression [74] with the maximum number of iterations until convergence set to $10,000$.

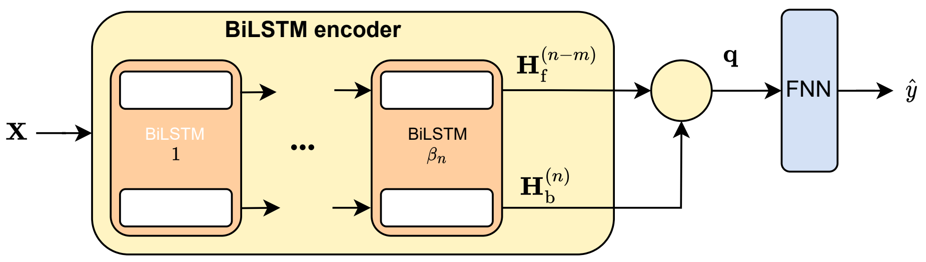

BiLSTM encoder
X
BiLSTM 1
...
BiLSTM β_n
$\mathbf{H}_\mathrm{f}^{(n-m)}$
$\mathbf{H}_\mathrm{b}^{(n)}$
q
FNN
$\hat{y}$

Figure 6.1: Network diagram for the proposed BiLSTM architecture. The input feature vector is fed into a set of stacked BiLSTMs. The last embeddings in both directions are concatenated followed by a fully connected layer for classification.

6.1.2. BiLSTM

RNNs have successfully been used in several acoustic classification tasks. Acoustic feature vectors are processed sequentially, at each step updating the network's internal hidden states which contain context-rich information and are available to the network at the next time step. This process allows complex temporal information within a set of acoustic features to be captured. In this work, we choose to use the LSTM architecture [50] due to its successful use in many related applications, such as automatic speech recognition and voice activity detection [75–79]. Concretely, we make use of a BiLSTM, which extends the LSTM architecture by processing the sequence in both forward and backward temporal directions. Initial informal tests using various model size configurations allowed us to conclude that the BiLSTM architecture consistently outperforms its LSTM counterpart for TB classification.

A high-level diagram of the proposed network is shown in Figure 6.1. First, a sequence of acoustic feature vectors $\boldsymbol{X} \in \mathbb{R}^{d_x \times n}$ where n is the number of frames and d_x is the dimensionality of the acoustic feature vector is fed into a BiLSTM encoder. The BiLSTM encoder contains β_n stacked BiLSTM cells, each of which has a hidden state size β_h whereby the final outputs due to both directions $\boldsymbol{H}_f^{(n-m)} \in \mathbb{R}^{\beta_h}$ and $\boldsymbol{H}_b^{(n)} \in \mathbb{R}^{\beta_h}$ are concatenated to form the vector $\mathbf{q} \in \mathbb{R}^{2\cdot\beta_h}$, where m is the number of padded frames and $\boldsymbol{H}_f \in \mathbb{R}^{\beta_h \times n}$ and $\boldsymbol{H}_b \in \mathbb{R}^{\beta_h \times n}$ are the output sequences of the last BiLSTM encoder in the forward and backward directions respectively. The formation of $\mathbf{q}$ for a BiLSTM encoder with only one BiLSTM cell is further illustrated in Figure 6.2. This latent representation is then passed through a small feed-forward network with a 32-dimensional hidden layer and ReLU activation followed by a two-dimensional output layer and softmax. We include dropout before the first linear layer with a probability of 0.5. The size of this fully connected network was determined through informal observations, adjustments of which are not investigated further.

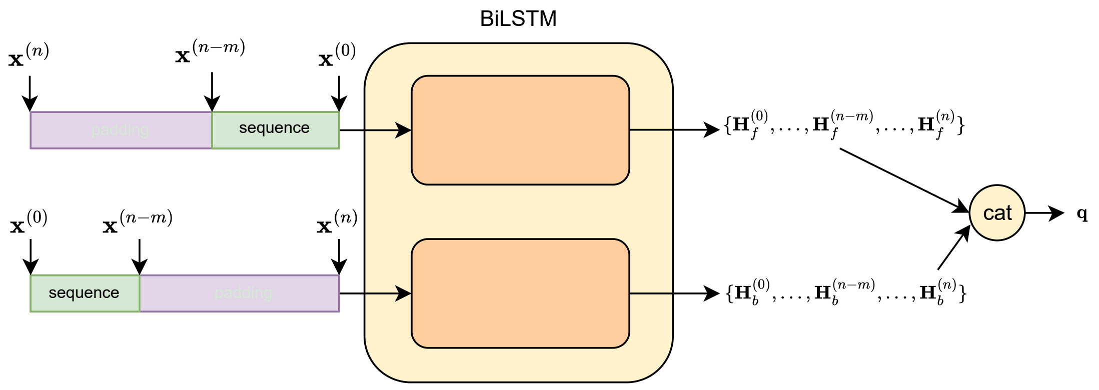


Figure 6.2: Illustration of an input sequence $\boldsymbol{X}$ of length n with m padded samples being fed into a single BiLSTM layer and the formation of the single vector $\mathbf{q}$. The last output of the forward LSTM that does not correspond to a padded input $\boldsymbol{H}_f^{(n-m)}$ is concatenated with the last output of the backward cell $\boldsymbol{H}_b^{(n)}$, which because it processes the padded elements fist, may include information extracted over the padded region.

Hyper-parameters

When designing the BiSLTM architecture, we investigate the trade-off between model complexity and performance. An increase in model complexity, for example by adding BiLSTM layers, may allow more complex temporal relationships to be captured, but also increases the number of model parameters, which could make the network harder to train or lead to over-fitting. As such, we conduct a grid search between two parameters which directly influence model complexity: the number of BiLSTM layers β_n and β_h. Whereby the search space for β_n is constrained to $\{1, 2, 3\}$ and β_h to $\{32, 64, 128, 256\}$.

Optimisation

We use the RMSProp optimiser [1] with a weight decay of 0.0001. We perform a learning rate sweep at the beginning of training to find the optimal learning rate.

6.1.3. BiLSTM-Attention

The development of the attention mechanism [51] has revolutionised deep learning research. With a focus on acoustic classification, attention-based architectures achieve near state-of-the-art results on tasks such as the Google speech commands dataset [45, 46]. In addition, the intuitive nature of the architecture allows for analysis of what the network is learning, reducing the black-box notion commonly associated with deep learning. The concept of attention is explained in detail in Chapter 3.

We develop an attention-based model by incorporating an attention layer in the previously described BiLSTM architecture. Instead of passing $\mathbf{q}$ directly to the fully

[1]Informal experiments showed RMSProp to outperform Adam for recurrent networks.

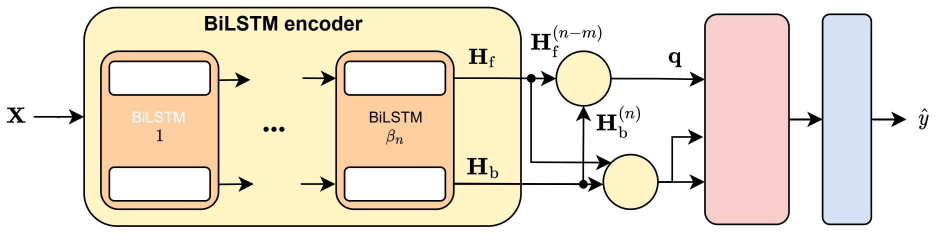


Figure 6.3: Network diagram for the proposed BiLSTM-Attention architecture. The input feature vector is fed into a set of stacked BiLSTMs. The last forward and backward representations in both directions are concatenated to form a query vector $\mathbf{q}$ which is parsed to an attention layer together with keys and values from the concatenated sequence of forward and backward representations ($\boldsymbol{H}_\mathrm{f}$ and $\boldsymbol{H}_\mathrm{b}$ respectively) outputted by the BiLSTM encoder. The attention mechanism outputs a single embedding which is the weighted sum of $\boldsymbol{H}$ by attention score relative to $\mathbf{q}$. This is then parsed to a fully connected network for classification.

connected network as is shown in Figure 6.1, we use a single head attention layer to weight the BiLSTM encoders final layer output embedding sequence $\boldsymbol{H} \in \mathbb{R}^{n \times 2 \cdot \beta_h}$ (where n is the length of the sequence) which is the concatenation of the BiLSTM encoder outputs in both the forward $\boldsymbol{H}_\mathrm{f}$ and backward $\boldsymbol{H}_\mathrm{b}$ directions as in [45, 46]. Importantly, attention is masked (forced to zero) for padded regions. We use $\mathbf{q}$ as a query vector and $\boldsymbol{H}$ as the keys and values, as illustrated in the high-level network diagram in Figure 6.3. This results in a single latent vector $\boldsymbol{e} \in \mathbb{R}^{\boldsymbol{2 \cdot \beta_h}}$ that captures information only from the temporal regions most relevant for classification, and suppress information from unimportant regions in time. This latent vector is then fed into the same fully connected network described for the previous BiLSTM architecture.

Hyper-parameters

We perform the same hyper-parameter optimisation followed for the BiLSTM architecture.

Optimisation

We use the RMSProp optimiser [2] with a weight decay of 0.0001. We perform a learning rate sweep at the beginning of training to find the optimal learning rate.

6.1.4. CNN

Owing to their ability to extract robust feature representations, CNNs have been widely used to perform a variety of tasks including image classification, object detection, image segmentation, and even speech processing [65–67, 80]. Moreover, as highlighted in Chapter 2, much related work has leveraged CNNs for cough classification. Therefore, we investigate their application to TB cough classification.

[2]Informal experiments showed RMSProp to outperform Adam for recurrent networks.

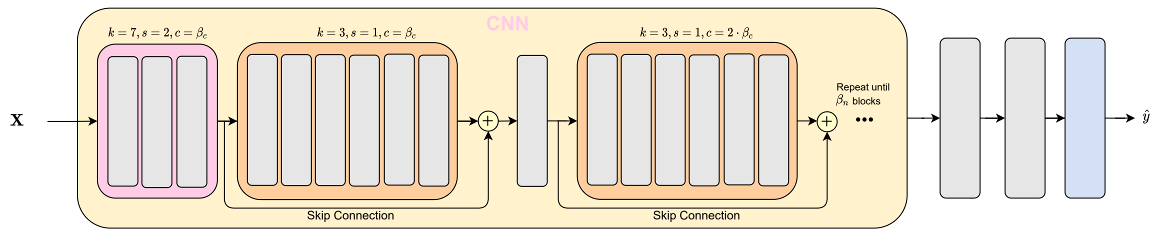

CNN
$k = 7, s = 2, c = \beta_c$
$k = 3, s = 1, c = \beta_c$
$k = 3, s = 1, c = 2 \cdot \beta_c$
Repeat until β_n blocks
Skip Connection
Skip Connection
$\mathbf{x}$
$\hat{y}$

Figure 6.4: Network diagram for the proposed CNN-based architecture. Each convolutional block is highlighted with a solid colour, where pink denotes the input block, and orange the convolutional blocks which will be stacked. Note k, s and c above each block denote the kernel size, stride, and the number of filters/channels respectively.

We construct a CNN architecture similar in structure to ResNet [4] which includes skip connections between stacked convolutional blocks with a focus on a reduced parameter count. We present our architecture in Figure 6.4. The designed network expects a fixed dimensional input of $\boldsymbol{X} \in \mathbb{R}^{1\times224\times224}$, chosen to be consistent with ResNet. However, the acoustic feature vectors used to represent cough are variable in length. To ensure all elements in a batch are the correct dimension, we simply zero pad along the time axis of the acoustic feature vector such that its length (or the number of frames) is 224 and then stretch the frequency axis to a height of 224 through linear interpolation. When the sequence of acoustic feature vectors is longer than 224, a simple energy detector is used. A 224 frame window is slid over the sequence with a frame skip of 1, whereby the window with the highest RMS energy (assumed to have the most cough information) is selected. After this pre-processing, $\boldsymbol{X}$ is passed through a convolution layer with stride 2 and kernel size 7 to capture initial contextual information (highlighted by pink in Figure 6.4) and compress the feature space. This is then followed by β_d convolutional blocks (highlighted by orange in Figure 6.4). A depth-wise expansion layer, which is a convolutional layer with c 1×1 kernels where c is the new channel dimension, is used to upsample the channel dimension to twice as many as the previous layer followed by three convolutional layers with stride 1 and 3×3 kernels with skip connections between the input of the first layer and output of the last. This is then followed by batch normalisation, max pooling, and dropout (fixed probability of 0.2). ReLU activations are used after each convolutional operation. To perform classification, we pool over the channel dimension of the final output activation map and flatten the resulting two-dimensional feature space such that it can be passed to a single layer fully connected layer for classification (two neurons and softmax).

Hyper-parameters

Since the number of filters required to extract the latent feature representations suitable for TB classification and how complex these latent representations should be is unknown, we set both the number of initial filters β_c and depth of the convolutional network β_d as hyper-parameters, and perform a grid-search between β_c and β_d from $\{8, 16, 32\}$ and

$\{3, 4, 5\}$ respectively. Note the complexity of features extracted scales with the depth of the network.

Optimisation

We use the Adam optimiser with a weight decay of 0.0001. We perform a learning rate sweep at the beginning of training to find the optimal learning rate.

6.1.5. Transformer

Transformers have been used successfully in a range of deep learning applications, in particular speech processing [72]. This is owed to their ability to capture context-rich, latent representations with a broad temporal range, a task that is much harder for RNNs. As such, we investigate the use of the basic transformer architecture in this work. We refer the reader to Chapter 3 for terminology and concepts related to attention and transformers, as this section will not reintroduce them for the sake of brevity (including positional encodings and encoders).

Our designed architecture takes as input a sequence of acoustic feature vectors $\boldsymbol{X} \in \mathbb{R}^{d_x \times n}$ (described in previous sections). First, $\boldsymbol{X}$ is linearly projected by a single point-wise feed-forward layer (i.e. the same feed-forward layer is applied to each element in the sequence) which has the same number of neurons as the feature dimensionality expected by the transformer β_d. A learnable classification token $\boldsymbol{x}_{\text{cls}}$ is then appended to the start of the resulting sequence which is then positionally encoded (the method of which will be detailed subsequently). This sequence is then fed into a series stack of transformer encoder blocks. Importantly, attention is masked for regions corresponding to padded inputs. The first embedding of the last encoder's output sequence, which corresponds to $\boldsymbol{x}_{\text{cls}}$, is then fed into a single fully connected 2-dimensional layer followed by softmax for classification. The general architecture is depicted in Figure 6.5. We experiment with an assortment of model configurations, which will be described subsequently.

Hyper-parameters

There are several parameters to consider when designing transformer architectures. Typically, performance is a function of width (number of attention heads β_{ah}), depth (number of encoder blocks β_{e}), and general model size (latent representation size β_{d} and each encoder's feed-forward size). Whilst the common trend in high-resource applications is to maximise all of these factors, this will likely inhibit learning in our low-resource task. As such, we perform a grid search on these parameters, with $\beta_{\text{ah}} = \{4, 8\}$, $\beta_{\text{e}} = \{4, 8\}$ and $\beta_{\text{d}} = \{128, 256\}$. We set the size of the point-wise feedforward layer in each encoder block to $2 \cdot \beta_{\text{d}}$ as is common in transformer applications [51, 64, 71]. This relative increase in the size of this layer to β_{d} is motivated by the fact that it is applied to every element in the

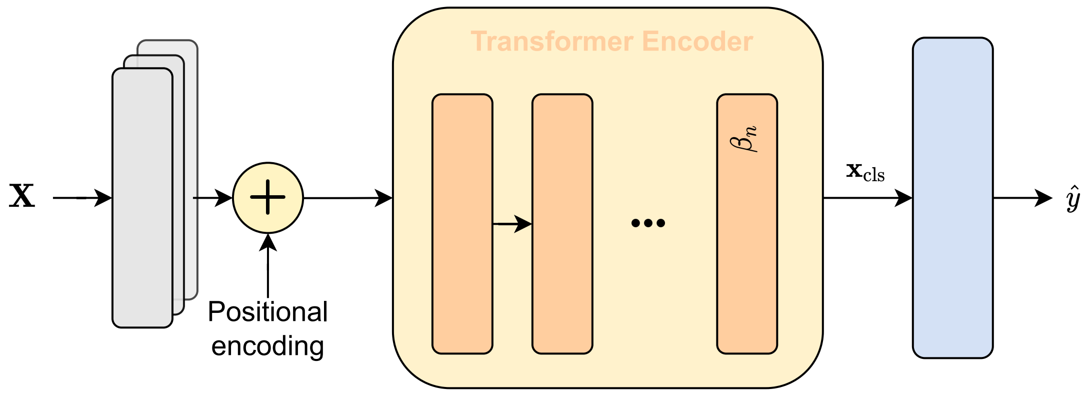


Figure 6.5: High-level diagram for the proposed transformer architecture. $\boldsymbol{X}$ is up-sampled in the feature axis, with a single linear layer (same layer used for each element in the sequence) before being positionally encoded. This in then fed through β_n transformer encoding modules, after which the CLS token is used for classification.

latent sequence outputted by the multi-head attention module, and as such needs to be able to transform a diverse range of latent representations.

In addition to architectural hyper-parameters, we also consider two different methods of positional encoding, namely the fixed sinusoidal embeddings described in Chapter 3, and a set of learnable positional embeddings $\boldsymbol{P} \in \mathbb{R}^{\beta_t \times \beta_\text{d}}$ as described in [81] similar to that used in AST, where β_t is the longest sequence seen during training. In essence, this allows the network to potentially learn more complex positional relationships between elements in the sequence, with the trade-off of not generalising to sequences longer than those seen during training.

Optimisation

We use the Adam optimiser with a larger weight decay than that used for other architectures of 0.01. This is motivated by the general observations in the literature, whereby larger weight regularisation tends to be used when training transformers [6, 42, 51, 64, 71]. We perform a learning rate sweep at the beginning of training to find the optimal learning rate.

6.1.6. ResNet

ResNet architectures have proven to transfer well to cough classification, in particular detecting COVID-19 as highlighted in Chapter 2. We investigate using both ResNet-50 (50 convolutional layers) and the smaller ResNet-18 (18 convolutional layers) as TB classifiers.

ResNet expects a fixed dimensional input of size $3 \times 224 \times 224$, which represents the three channels of an RGB image with a height and width of 224 pixels. Whilst it is common practice in computer vision to interpolate or resize data to match that expected by the model, this procedure would inhibit the learning of sensitive temporal information contained in the sequence of acoustic feature vectors, as different samples would be stretched or squeezed by varying amounts depending on how many frames were present in the original sequence. To circumvent this, we simply zero pad along the time axis of the acoustic feature vector such that its length (or the number of frames) is 224 and then stretch the frequency axis to a height of 224 through linear interpolation. When the sequence of acoustic feature vectors is longer than 224, a simple energy detector is used. A 224 frame window is slid over the sequence with a frame skip of one, whereby the window with the most cough information (highest RMS energy) is selected. The resulting fixed-dimensional acoustic feature vector is then stacked three times along a new dimension to represent the three RGB channels. To perform classification, the latent 3-dimensional feature map outputted by the last convolutional layer is averaged over its width and height, as is performed in the original implementation of ResNet [4]. The resulting embedding (the same dimension as the number of channels in the aforementioned feature map) is then fed into a fully connected layer for classification. The size of this latent vector is 512 and 2048 for ResNet-18 and ResNet-50 respectively.

Hyper-parameters

We investigate not updating parameters (freezing) from portions of the pre-trained architecture (also referred to as a backbone) during training. This is motivated by the hypothesis that most parameters learnt during pre-training should be generalised, and as such should not all have to be fine-tuned to the task at hand (and hence mitigate the potential for over-fitting). For example, the early layers in ResNet have been shown to learn very low-level patterns (edges, textures, lines etc.), which would be useful for most fine-tuning tasks, and hence should not need to be retrained [82]. We freeze/unfreeze different portions of the backbone to investigate how generalised the pre-trained architecture parameters are. Namely, we experiment with freezing the entire backbone, freezing the entire backbone except for the last convolutional block, and finally unfreezing the entire backbone and allowing the loss to back-propagate through the entire model.

Optimisation

We use the Adam optimiser with a weight decay of 0.0001. We perform a learning rate sweep at the beginning of training to find the optimal learning rate.

6.1.7. Audio Spectrogram Transformer

Motivated by AST's ability substantially improve upon audio classification benchmarks, we investigate fine-tuning it for TB cough classification. The setup of AST remains the same as that previously described in Chapter 5, except for the replacement of the final linear layer which has 527 output classes with a 2-dimensional linear layer suitable for our binary classification task.

Hyper-parameters

We do not investigate any hyper-parameters when training AST and consider only the exact implementation used in [5], which uses 128-dimensional mel-spectrograms as acoustic feature vectors.

Optimisation

We use the Adam optimiser with a weight decay of 0.01. We perform a learning rate sweep at the beginning of training to find the optimal learning rate.

6.1.8. wav2vec2.0

Motivated by wav2vec2.0's state-of-the-art ability to extract complex contextualised acoustic representations, we use the smaller "base" variant as a pre-trained backbone for TB classification and experiment with fine-tuning it for TB cough classification.

Since wav2vec2.0 was trained on 16kHz audio, we down-sample our cough audio to match this. In [6] it was shown that using a decoder on top of a wav2vec2.0 backbone can help convert the extracted speech representations into speech command features suitable for keyword spotting, and results in improved performance in comparison to using a simple linear classification layer which takes the last element in the sequence outputted by the transformer as input. We experiment with both techniques. Importantly, all input samples are padded or cropped to one second, masking attention for padded samples. This was performed for two reasons: padding all input samples in a batch to the longest sequence (as is done with the recurrent architectures) required too much memory for an adequate mini-batch size for training to converge, and secondly, the contextual embedding decoder used is designed to work with strictly one second long inputs. Cropping is performed by simple energy detection, whereby a one-second window (16,000 samples) is slid over the entire audio waveform with a hop length of one. For each window of audio, the root-mean-square (RMS) of its amplitudes is computed. The window with the highest RMS amplitude is used as the input to the network.

As in [6] our decoder is made up of two 1D convolutional layers. The first performs time compression with 112 kernels of size 25 and dilation of 2 designed to convert the 49 contextual embeddings $\boldsymbol{C} \in \mathbb{R}^{768\times 49}$ outputted by the backbone for the one-second-long input to $\boldsymbol{C}' \in \mathbb{R}^{112\times 1}$. The second convolutional layer applies point-wise convolutions (kernel of size 1), the output of which is fed to a fully connected network for classification. The complete structure of the decoder is shown in Figure 6.6, whereby each convolutional block makes use of batch normalisation and ReLU activation functions.

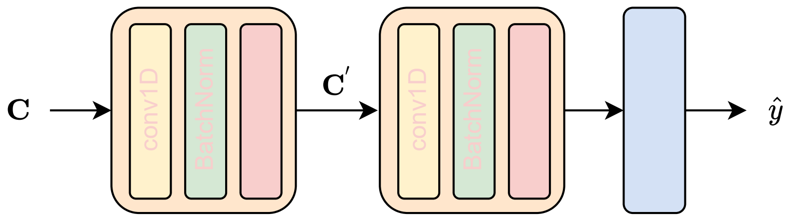


Figure 6.6: wav2vec2.0 decoder structure used in [6]. Context representations are fed through two 1D convolutional blocks, and then a fully connected layer for classification.

When using a simple linear classification head directly on top of the backbone, we arbitrarily use the last element of the sequence $\boldsymbol{C}$ described previously. In theory, any element should suffice, as after passing through all 12 transformer encoder layers, the relevant sequence context for classification should be embedded in each output embedding.

Hyper-parameters

Whilst limited architecture modifications can be made, we experiment with freezing different layers of the backbone during training to investigate its effect in reducing over-fitting and subsequent improvement of performance. This includes freezing the CNN feature extractor and the entire backbone, although we only perform the latter when the model has been pre-trained on the auxiliary cough detection task. Some informal preliminary experimentation was conducted concerning using earlier transformer layer outputs instead of the final latent embeddings, but this always resulted in reduced performance.

Optimisation

We use the Adam optimiser with a weight decay of 0.01. We perform a learning rate sweep at the beginning of training to find the optimal learning rate.

6.2. Pre-training

It is well documented in the literature that pre-training large neural networks on related tasks before fine-tuning them on the primary task can lead to improved performance. Pre-training networks as cough classifiers, that is classifying if an audio signal contains a cough or not, has shown to improve the performance of COVID-19 cough classifiers [19, 20, 34]. Motivated by this, we investigate the application of such pre-training techniques to TB cough classification and use a similar setup to that described in [34]. Concretely, we use a self-constructed dataset titled “Cough or Not” as described in Chapter 4.

As highlighted in previous sections, extensive hyper-parameter searches are conducted for many investigated architectures. It is not feasible to pre-train each configuration due to computational constraints, and as such we only investigate pre-training the configuration which performed best during development. All networks are trained for 30 epochs, with a

Table 6.1: Dimensionalities considered for various acoustic feature vectors.

Acoustic Feature	Dimensionalities considered
LFBE	$80, 128, 180$
Mel-spectrogram	$80, 128, 180$
MFCCs	$13 \times 3, 26 \times 3, 39 \times 3$

decaying learning rate initialised by a learning rate sweep. After each epoch, classification accuracy is evaluated on a held-out development set, after which model weights are stored if the performance metric has increased. Due to the presence of some substantially long recordings in the constructed "Cough or Not" dataset, memory constraints were reached for larger models. To circumvent this, we use the same simple energy detector used for the CNN, ResNet and wav2vec2.0 architectures. A fixed-length window is slid over the sequence of acoustic feature vectors, selecting the sub-set of acoustic feature vectors that correspond to the window with the highest RMS energy. The window lengths for all other architectures which did not already use an energy detector (BiLSTM, BiLSTM-Att, transformer, and AST) were set to 600 frames.

After pre-training, the "Cough or Not" classifier's weights were used to initialise a corresponding TB classifier, with the exception of the final fully connected layer (responsible for outputting class probabilities) which was discarded. The same training procedure is followed as with the non-pre-trained (base) networks. Importantly, when discussing experimental results, already pre-trained architectures (e.g. ResNet) will still be referred to as base models if they have not yet been further fine-tuned on the "Cough or Not" task.

6.3. Acoustic features

We previously described LFB energies, mel-spectrograms, and MFCCs in Chapter 3. All three of these acoustic feature vectors are experimented with in this work. However, due to the computational complexity of searching for optimal acoustic features for each architecture in addition to each specific architecture's own extensive hyper-parameter search, we use the results of an initial study to identify the class of acoustic representations that results in the best performance across a few models and make the naive assumption that this will also be a good choice in the remaining experiments. In this initial investigation, we also explore the impact of the dimensionality of each acoustic feature vector on model performance. Three sizes for each acoustic feature are experimented with and are highlighted in Table 6.1. Note the multiplication factor of three for each MFCC dimension, which accounts for the inclusion of velocity and acceleration coefficients. The range of feature vector dimensionalities is based on those explored in [2] and [3] as well as other related cough classification literature described in Chapter 2.

We use a fixed window and hop length of 2048 and 512 respectively. This is motivated by parameters found to be ideal in [2, 3] in addition to [83] which all showed through rigorous experimentation that larger windows are more beneficial for capturing frequency information relevant for both cough classification and detection. Due to the already large computational extent of our experiments we take advantage of results presented in these related studies and do not investigate tuning these parameters, however, we acknowledge that a thorough search for optimal parameters in this specific application may improve results.

6.3.1. Data augmentation

We experimented with two data augmentation techniques: SpecAugment [7], and speed-perturbation [84]. Whilst it is often common to augment audio datasets by the addition of environmental noises, we do not explore this in this work due to the already noisy nature of our dataset. We conduct an initial informal experiment comparing performance with the aforementioned augmentations on two deep classifiers (BiLSTM and BiLSTM-Attention) to conclude which augmentation techniques will be used in later work. We describe each augmentation in more detail subsequently.

SpecAugment

SpecAugment is a simple data-augmentation technique that applies time warping, frequency masking, and time masking directly to a sequence of acoustic feature representations [7]. Given a sequence of n acoustic feature vectors with d_x feature bins, time stretching is achieved by first randomly selecting a frame between $(W, n-W)$ to serve as an anchor point, where W is the time warp parameter. The anchor point is then either moved either left or right by some factor w sampled uniformly between $(-W, W)$, whilst either stretching or compressing the sequence of frames to the left and right of the anchor point as it moves to its new position through linear interpolation. This results in one half of the sequence resembling a speed-up of audio, and the other a slow-down. For example, if the 30th frame of a 50-frame spectrogram was selected, and a w of $+5$ (move right by 5) was sampled, the first 29 frames would be interpolated to a new length of 34, and the last 20 frames would be interpolated to a new length of 15. This concept is further reinforced in Figure 6.7, where we visualise this transformation for a base spectrogram (top), and the resulting warped spectrogram (bottom).

Both time and frequency masking are comparatively more simple. With respect to frequency masking, the width of the masking band (the number of consecutive bins to mask) f, is sampled uniformly between $[0, F)$, where F is the frequency masking parameter. Next, the bin f_0 from where the masking starts is sampled from $[0, d_x - f]$. Masking is applied to each element in the sequence of acoustic feature vectors. A similar process is followed for time masking, where the masking parameter T determines the maximum

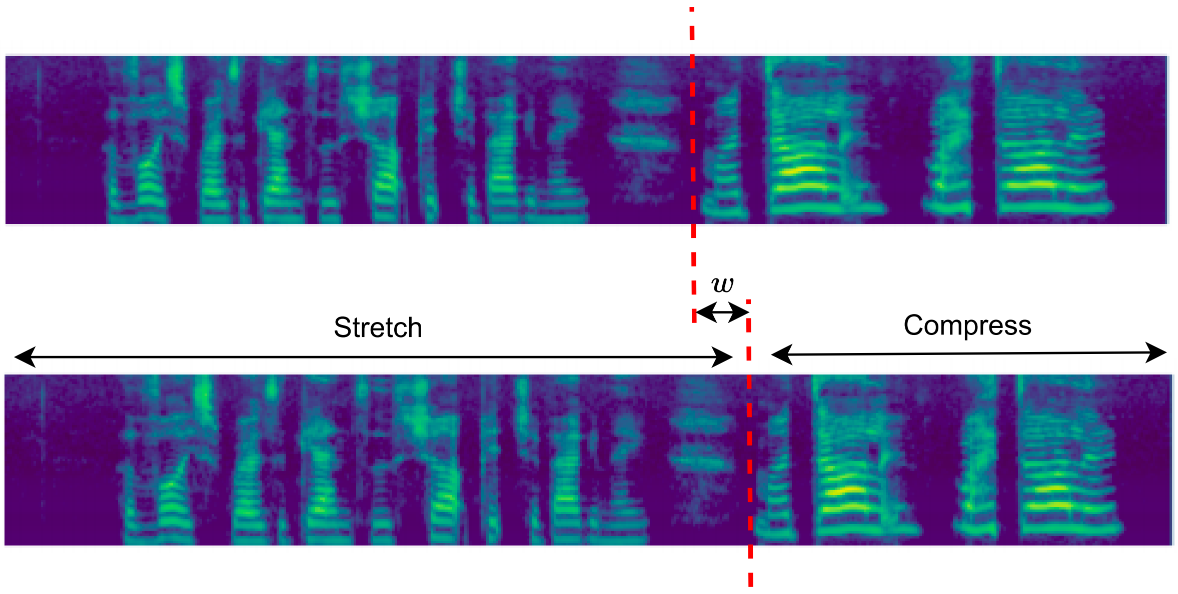


Figure 6.7: Illustration of time warping being applied to a spectrogram (top) and the resulting transformation (bottom). The dashed red line indicates the selected anchor point [7].

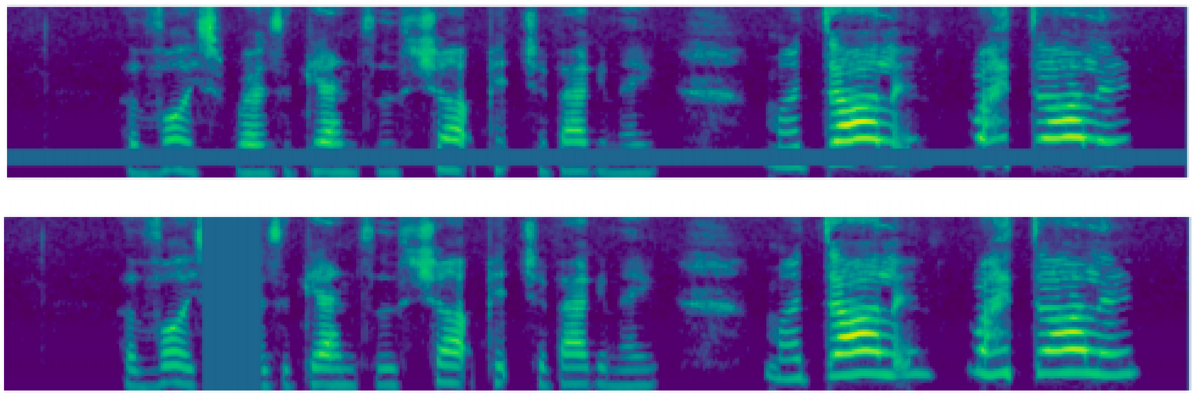


Figure 6.8: Example of frequency (top) and time (bottom) masking [7].

number of consecutive frames to be masked. Note that in time masking, the entire acoustic feature vector is masked. We provide an example for both frequency and time masking in Figure 6.8.

Since we are unsure as to where important information lies in a TB cough, and how nuanced the signal in time can be, we choose F and T to be only 5% of d_x and n, and only apply masking once. W is chosen to be 95% of n such that the time warp is sufficiently small so as to not destroy any temporal information. Whilst potentially more optimal parameters may be possible, due to computational constraints we could not investigate them further. We did however conduct informal experiments, which showed some degree of performance insensitivity around these values implying that they are within a good range.

Speed-perturbation

A simple yet effective audio augmentation technique is to create additional training data by simply speeding up or slowing-down original samples. In [84] such a technique was proposed for data augmentation for an ASR task, whereby an audio signal $x(t)$ is warped by some warping factor α to produce the signal $x(\alpha t)$. By analysing the Fourier transform of this perturbed signal, $\alpha^{-1}X(\alpha^{-1}f)$, we observe that this causes some shift in frequency information with respect to $X(f)$. In addition, the change in the duration of the signal results in a different number of acoustic feature frames from those derived for the original audio signal. This technique was applied to the training sets of ASR systems, which increased the dataset size by a factor of three by using warping factors $\alpha = \{0.9, 1, 1.1\}$ and resulted in a 4.3% reduction in word error rate on average.

In this study, we experiment with using these same warping factors as in [84] to augment training data. Whilst an investigation into the number and size of warping factors on performance may yield a more ideal configuration for this augmentation technique, we leave this for future study.

6.3.2. Sequential forward search

SFS has been shown to substantially improve classifier performance (in particular AUC) in the literature [2,3,8] and as such we investigate its application in our work. SFS is an algorithm used to determine the feature vector subset that results in optimal classifier performance by means of a greedy search that sequentially adds the individual feature to the feature subset that results in the greatest improvement in classification performance (based on the development set). This procedure is outlined by the pseudo-code below, where J is the criterion function that is being optimised. Owing to SFS's $O(N^2)$ complexity, it is infeasible to apply it to all investigated architectures. For this reason, we only conduct SFS for the optimal configuration of the best non-pre-trained architecture as presented in the subsequent chapter.

Algorithm 6.2: Sequential forward search feature selection.

```
Let the input feature set be: $X \leftarrow \{x_1, x_2, ..., x_d\}$
Create the empty sets: $Y \leftarrow \emptyset, Q \leftarrow \emptyset$
Let: $k \leftarrow 0$
while $k \neq d$ do
    $x^+ = \text{argmax } J(Y + x)$, where $x \in X - Y$
    $Q = Q + J(Y + x^+)$
    $Y = Y + x^+$
    $k = k + 1$
end while
Let the optimal feature set index be: $q \leftarrow \text{argmax } Q$
return $Y_{0:q}$, the optimal feature set, ranked
```

Table 6.2: Train and test set splits of the combined dataset for the three outer folds (fold 1 to 3 from left to right).

Set	TB	$\overline{\text{TB}}$	Total
Train	19	30	49
Test	9	16	25
Total	28	46	74

Set	TB	$\overline{\text{TB}}$	Total
Train	18	31	49
Test	10	15	25
Total	28	46	74

Set	TB	$\overline{\text{TB}}$	Total
Train	19	31	50
Test	9	15	24
Total	28	46	74

6.4. Datasets, evaluation and training

We now detail the general experimental setup used in this work. This includes details regarding the training, development, and testing of models. In addition, we explain the evaluation metrics which will be used to quantify model performance. Lastly, we summarise experiments that will be conducted, most of which have already been introduced in this chapter. Experiments were conducted in Python, and unless otherwise stated implemented with PyTorch and PyTorch Lightning.

6.4.1. Dataset splits

We divide the combined dataset, as described in Section 4.3, into three training set and test set partitions, whereby patients in one test do not appear in any other. This three-fold cross-validation (referred to as the outer folds) is in an effort to evaluate model robustness to unique sets of unseen patients, whereby experiments are repeated for each fold independently. Each of these train/test splits is shown in Table 6.2. Importantly, both the Brooklyn and Wallacedene datasets are represented equally in all splits. Furthermore, splits are performed patient-wise, ensuring that all coughs originating from the same patient are only present in one set, and we ensure a uniform distribution of TB and $\overline{\text{TB}}$ patients across splits.

Each of the three training sets is further divided into four inner folds (each consisting of its own train and development set) for development cross-validation using the same previously described procedure. Importantly, there is again no patient overlap between the development sets. Any given split which is being used to train and evaluate an individual model will typically rely on ≈ 38 patients for training and ≈ 12 for development. We highlight how these partitions were performed in Figure 6.9. Note the non-overlapping nature of both train/test and train/development splits, whereby each element in the dataset (patient) only appears in a test set once, whilst the same is true for inner-fold development sets. The precise contents of each split in terms of patient IDs are given in Appendix B. We refer to this procedure of utilising outer folds for testing and inner folds for development as nested k-fold cross-validation.

A keen eye will notice that patients used in the test set of, say, fold 1 for example, will

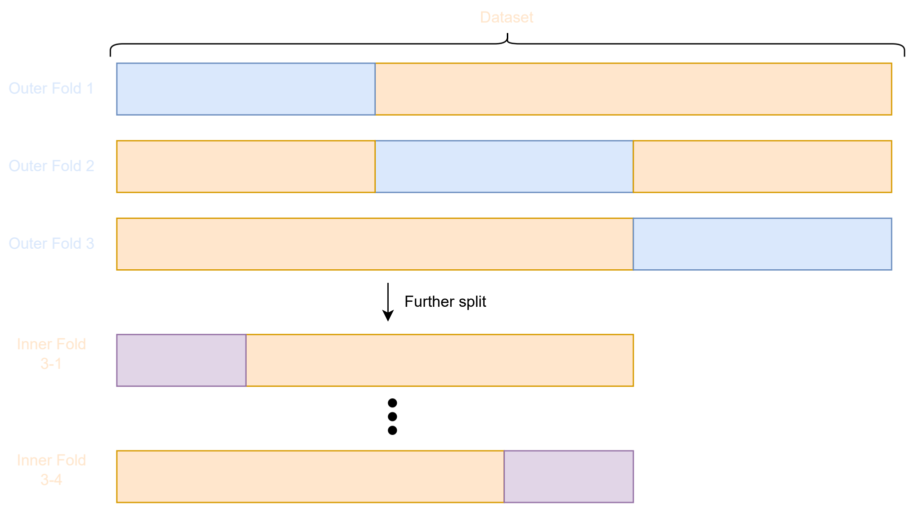


Figure 6.9: Visualisation of nested k-fold cross-validation used in this work. The dataset is divided into three non-overlapping subsets (or three outer folds). Each training set is further divided in four non-overlapping subsets (or inner folds). These are used to optimise architecture parameters for that specific outer fold. This is repeated for each inner and each outer fold.

partially appear (50% in fact) in the development sets of fold 3. This overlap across outer folds between test and development sets must be handled with care, and means decisions during development must be independent across other outer folds. This phenomenon is highlighted in more detail in Appendix C. With this in mind, it is important to state that these experiments were not initially designed with nested k-folds cross-validation in mind, and hence unwanted biases may have crept in. For example, when performing development in fold 1 (which was initially the only split used during experimentation) we might have observed a specific range of hyper-parameters performing better on the development set than others, which would have been used to inform decisions made when selecting which hyper-parameters to search for when conducting nested k-folds cross-validation.

Inner fold development

During hyper-parameter optimisation, for each iteration, we fix the set of hyper-parameters currently being investigated for all inner folds belonging to a specific outer fold and use the mean development scores to determine the optimal hyper-parameters for that given outer fold. In other words, we try to find a set of hyper-parameters that result in the best-generalised performance across all development folds for a given outer fold split, rather than trying to optimise each inner fold individually, which would almost certainly result in development set over-fitting, a phenomenon whereby model hyper-parameters are tuned to such a degree that that whilst optimising development set performance, they no longer generalise to other unseen data.

6.4.2. Performance Metrics

A variety of performance metrics are used to evaluate the effectiveness of the classifiers considered in work. These include sensitivity, specificity equal error rate (EER) and AUC. To better understand the metrics presented subsequently, it is first necessary to understand the concepts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) after selecting a decision threshold. TP and TN quantify the number of correctly classified samples with ground truth positive and negative labels respectively, whilst FN and FP quantify the number of incorrectly classified samples with ground truth positive and negative labels respectively. High TP and TN values indicate a classifier that can easily and reliably distinguish between two classes, whilst high FN and FP values indicate that the classifier is *confused*, and cannot accurately determine the class a data point belongs to. These are best described visually and are depicted through a confusion matrix for a binary classifier (predicts two classes, 1 or 0), in Figure 6.10. Typically each row in the confusion matrix is normalised by the total number of ground truth samples for the specific class it corresponds to.

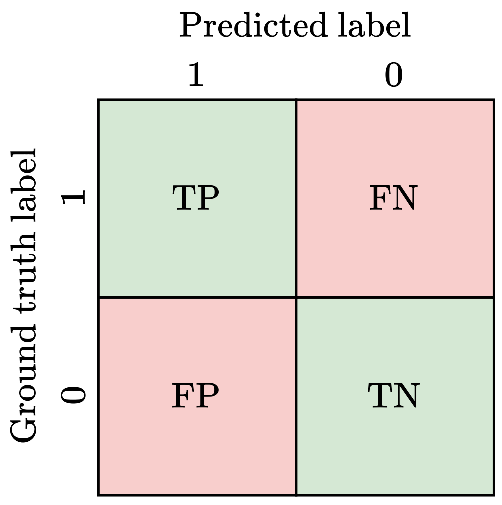


Figure 6.10: A confusion matrix depicting the ground truth and predicted labels for TP, TN, FP, and FN decisions. Cells highlighted in green indicate metrics where higher values indicate a better classifier, whereas red indicates metrics where lower values infer better performance.

Sensitivity

Sensitivity is a performance measure for binary classification systems which aims to partially mitigate the effects of class imbalance in performance metric computation by considering only the proportion of true positive samples correctly classified. This is shown in Equation 6.2, where TP + FN corresponds to the number of ground-truth positive samples in the evaluation set. By observing the equation, we see that a higher sensitivity indicates fewer FNs, and therefore that a classifier's positive predictions are more reliable. This is especially important for medical systems, whereby miss-classifying a patient who is sick can have severe consequences.

$$Sensitivity = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.2}$$

Specificity

Specificity accomplishes the same task as sensitivity, but for the negative class, by representing the proportion of negative samples correctly predicted. As shown in Equation 6.3, specificity is the measure of correctly predicted TN samples compared to the total number of ground-truth negative samples in the evaluation set ($\text{TN} + \text{FP}$). Thus, higher specificity indicates a classifier that will result in fewer FPs. This, in the context of a medical system, corresponds to a reduction in the number of people being sent for further testing but who are not in fact ill.

$$Specificity = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{6.3}$$

AUC-ROC

The ROC curve is determined by plotting the true positive rate (TPR) which is equivalent to the sensitivity, on the vertical against the false positive rate (FPR) shown in Equation 6.4, on the horizontal as the decision threshold is varied. Note FPR is in fact equivalent to $1 - \text{specificity}$. The ROC curve is a useful tool since the trade-off between sensitivity and specificity can be easily visualised. A curve that passes through the upper left corner of the plot indicates a perfect classifier, whereas a curve that lies along the $45°$ diagonal represents a classifier making random decisions.

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} \tag{6.4}$$

While ROC is a useful visualisation tool, it can become difficult to use effectively when comparing classifiers of similar performance. As such, it is useful to compare classifiers by considering the area under the ROC curve (AUC). In doing so, AUC characterises the classifier across all operating points. Note, $\text{AUC} \in [0, 1]$ where higher values indicate a better classifier and values approaching 0.5 indicate performance similar to a random classifier. An example is provided in Figure 6.11 for two classifier systems, A and B. In this example, system A is the better-performing classifier, and therefore its ROC curve lies above that of system B.

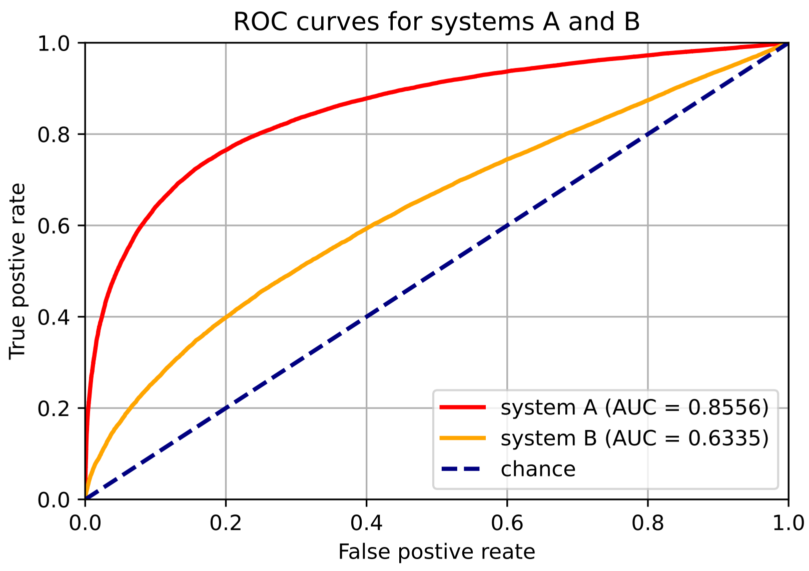


Figure 6.11: ROC curves for two systems: A and B.

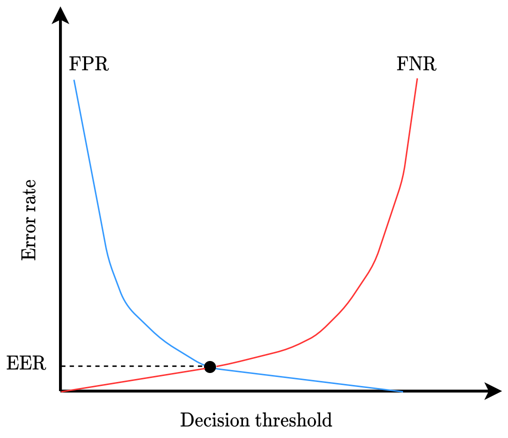


Figure 6.12: Representation of FPR and FNR as the decision threshold is increased. The error rate where FPR and FNR cross is labelled the EER.

Equal error rate

EER is the error rate at the decision threshold where the false negative rate (FNR) equals the FPR (recall, FPR is equivalent to $1 -$ specificity), where the FNR is simply $1-$sensitivity. As the decision threshold of a system is increased, the FPR will naturally decrease while the FNR will increase. For a good classifier, the FPR should decrease rapidly, and thus FNR and FPR should cross over at a low error rate, depicted in Figure 6.12. Therefore, a lower EER indicates a better classifier. In this work, in addition to its use a performance measure, we also use the EER to determine optimal decision thresholds during development as is done in [2] and [3].

Table 6.3: Batch sizes used when training various architectures.

Architecture	Batch size
BiSLTM	128
BiSLTM-Att	128
CNN	64
Transformer	64
ResNet	64
wav2vec2.0	32
AST	24

6.4.3. Training

Except for LR, which follows the standard `scikit-learn` training recipe [74], all models are trained for 15 epochs. This was always found to be sufficient for the binary cross-entropy training loss to converge. Moreover, to account for class imbalance during training we use weighted cross-entropy, whereby class weights are their respective inverse proportion of the training set. The details for each optimiser used for a specific architecture can be found in Section 6.1. It was found empirically that larger batch sizes tended to result in better performance until a batch size of 128 was reached. Thus, for each network, we try to achieve a batch size as close to this number in increments of 2^n where n is an integer. We tabulate these batch sizes for each architecture in Table 6.3. All models were trained on the same system running Ubuntu 20.04 with 16GB of RAM and an Nvidia GTX 1070Ti GPU.

6.4.4. Model selection and evaluation

After training is completed for all four inner folds as described in Section 6.4.1, the mean development AUC over the four respective inner folds is computed for each epoch. The models from the epoch with the highest mean AUC are selected as optimal. For a given architecture and outer fold, the configuration with the highest mean development AUC is selected to be evaluated on the corresponding outer fold test set. Importantly, this selection is independent for each outer fold and as such the optimal configuration of a specific architecture is not necessarily the same for each.

The prediction ensembling process followed when evaluating optimal architectures on the test set to determine a patients probability of having TB $\hat{\hat{y}}$ is described by Equation 6.5, where n is a particular inner development fold and $\hat{y}_n$ is the estimated probability of a patient having TB as predicted by the model for that fold. To determine $\hat{y}_n$ for a particular patient, we take the mean probability predicted for each of their C coughs being a TB positive cough $\hat{z}$ as shown in Equation 6.6 as in [2].

$$\hat{\hat{y}} = \frac{1}{4}\sum_{n=1}^{4} \hat{y}_n \tag{6.5}$$

$$\hat{y}_n = \frac{1}{C}\sum_{c=1}^{C} \hat{z}_c \tag{6.6}$$

To determine the optimal decision threshold for $\hat{\hat{y}}$, the mean of the decision thresholds γ_n that result in the EER for each inner fold is determined:

$$\bar{\gamma} = \frac{1}{4}\sum_{n=1}^{4} \gamma_n \tag{6.7}$$

Hence the optimal decision threshold was chosen based on the EER. We note that it might be possible to improve performance if a strategy that chooses this threshold to optimise, for example, sensitivity and specificity, is adopted. However, we leave this investigation for future work.

6.5. Summary

In this chapter, we detailed experiments conducted to investigate the application of deep learning -based classifier architectures to TB cough classification. This included recurrent architectures (BiLSTM with and without attention), convolutional-based models (CNN, ResNet-18, and ResNet-50) and transformers (including our own small-scale implementation, AST, and wav2vec2.0). We then detailed a pre-training scheme, and its application to investigate its effect on model performance. Next, the hyper-parameters for the acoustic feature representations investigated in this work were detailed, followed by data augmentation techniques, and a greedy feature selection algorithm SFS. Lastly, an in-depth description of the experimental setup used in this work was provided, which included details regarding dataset splits, performance metrics, training, model selection and evaluation. In the next chapter, we will present the results of the described experiments.

Chapter 7

Experimental Results

In the previous chapter, we described architectures and general experimental setup with regard to the investigation into the application of deep learning for TB cough classification. In this chapter, we first present the results of experiments conducted to find optimal meta-parameters (data augmentations and acoustic features). Then, we present the performance of each investigated architecture individually in comparison to the baseline system with and without pre-training and describe the configurations found optimal across outer folds. We then summarise and discuss these results, and present our observations and findings.

7.1. Optimal meta-parameters

First, we present the performance impact of the two data augmentation techniques on the development results on the first outer-fold for an 80-dimensional mel-spectrogram in Table 7.1. Note that this does not strictly follow the correct approach for nested k-fold cross-validation outlined in Appendix C, as we do not perform this investigation for each outer fold as it was too computationally expensive. We experimented with applying the two augmentations sequentially. That is, first speed-perturbation, and then SpecAugment to the perturbed data. As such, the performance of SpecAugment is relative to that reported for speed-perturbation. We note that only speed-perturbation increases AUC, and therefore is the only augmentation used in all subsequent work. We suspect that the degradation in performance when applying SpecAugment was a result of the models being unable to learn reliable relations between frequency bins carrying the TB signal due to the stochastic nature of the masking process, but this is speculation.

Table 7.1: Mean development AUC for the first outer-fold for the two investigated augmentation techniques. Namely, speed-perturbation (S-P) and SpecAugment.

Model	Augmentation		
	None	S-P	S-P + SpecAugment
BiLSTM	0.7410	**0.7450**	0.7196
BiLSTM-Att	0.7990	**0.8460**	0.8268

Next, we present experiments using the same outer fold for various acoustic feature configurations in Table 7.2. Again, although this does not strictly follow the correct approach for nested k-fold cross-validation outlined in Appendix C, it was too computationally expensive to perform this experimentation for all outer folds, and all architectures. Overall, mel-spectrograms are superior to both LFB energies and MFFCs, with the latter performing substantially worse. This is likely a result of the inclusion of the Brooklyn dataset in our combined dataset, for which [2] also found MFCCs to perform poorly (albeit that they did not consider the higher dimensional MFFCs considered here). Since no clear conclusions can be made as to the optimal dimensionality of the mel-spectrogram used, we leave this as a hyper-parameter in the work that follows.

Table 7.2: Mean and standard deviation of AUC and the decision thresholds ($\bar{\gamma}$) during 4-fold cross-validation.

Model	mel-spectrogram			LFB energies			MFCCs		
	80	128	180	80	128	180	13	26	39
LR (*baseline*) [2, 3]	0.5960	0.6850	**0.7070**	0.6290	0.676	0.6600	0.5450	0.553	0.5820
BiLSTM	0.7450	**0.7770**	0.7470	0.6200	0.727	0.7570	0.6260	0.6590	0.6080
BiLSTM-Att	**0.8460**	0.7990	0.8110	0.7410	0.7440	0.7520	0.6210	0.5800	0.6590

7.2. Classifier performance

In this section, we detail the performance of each optimised classifier (with and without pre-training) compared to the baseline system separately. When doing so, we present the mean and standard deviation of performance metrics across all three outer fold test sets. After analysing each classifier's performance, we summarise the optimal configurations for each outer fold and draw conclusions about the generalisability of the investigated architectural hyper-parameters. This detailing of individual performance with respect to the baseline is followed by a summary of all classifiers' performance, whereby general observations and trends are highlighted.

7.2.1. BiLSTM

We present the mean and standard deviation of the test set performance of the BiLSTM architecture across the three outer folds in Table 7.3. Both the base and pre-trained models outperform the LR baseline. A large increase in AUC and decrease in EER is observed when pre-training, indicating an improvement in the model's overall ability to distinguish between TB and $\overline{\text{TB}}$. In conjunction, a decrease in the standard deviation of performance metrics (Notably in EER and AUC), indicates better generalisation across folds. However, the substantial decrease in mean sensitivity indicates a sub-optimal decision threshold was chosen. During development, there was a large discrepancy in optimal architectural parameters (including feature dimensionality) for each fold. With the optimal hidden state sizes of 32, 64, and 32 and BiLSTM layers of 2, 1, and 1 being selected for the respective folds, whilst all investigated feature dimensionalities (80, 128 and 180) were found optimal for only one fold.

Table 7.3: BiLSTM test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2,3]	0.4741 ± 0.2983	0.7417 ± 0.0988	0.3694 ± 0.0275	0.7202 ± 0.0405
BiLSTM (base)	$\mathbf{0.6778 \pm 0.0157}$	0.6278 ± 0.0906	0.3264 ± 0.0098	0.7497 ± 0.0815
BiLSTM (pt)	0.3556 ± 0.1841	$\mathbf{0.9583 \pm 0.0589}$	$\mathbf{0.1305 \pm 0.0039}$	$\mathbf{0.7874 \pm 0.0268}$

7.2.2. BiLSTM-Attention

The mean and standard deviation of test set performance for all folds are presented in Table 7.4. We do not observe substantial performance increases with the BiLSTM-Attention architecture in comparison to the baseline system, and in some cases observe a decline in performance (specificity and EER). However, when pre-trained, we observe most metrics improving on the baseline system except for sensitivity, indicating non-optimal decision thresholds were selected. Larger standard deviations for EER (and AUC for the base system) are observed compared to the baseline system indicating poorer generalisation. Like the plain BiLSTM architecture, there was a large degree of variation between the optimal architectures found for each fold during development, with the hidden state sizes being found optimal for each fold being 128 or 80 with either 1 or 2 BilSTM layers.

Table 7.4: BiLSTM-Attention test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2,3]	0.4741 ± 0.2983	0.7417 ± 0.0988	0.3694 ± 0.0275	0.7202 ± 0.0405
BiLSTM-Att (base)	$\mathbf{0.6481 \pm 0.1142}$	0.6069 ± 0.0631	0.3708 ± 0.0683	0.7216 ± 0.0759
BiLSTM-Att (pt)	0.3185 ± 0.0733	$\mathbf{0.91389 \pm 0.0610}$	$\mathbf{0.2638 \pm 0.0982}$	$\mathbf{0.7387 \pm 0.0378}$

7.2.3. CNN

Test set performance for the CNN-based architecture is presented in Table 7.5. Except for sensitivity, the base CNN model performed substantially worse than the baseline system, with notable relative reductions of 17.7% in AUC and 39.6% in specificity. When pre-training, we observe the AUC returning closer to the baseline, however, see the same trend in previous architectures whereby the sensitivity drops substantially. Large standard deviations are observed for the base CNN architecture, which reduces when pre-training, albeit still sometimes higher than the baseline (EER, AUC), indicating poor generalisation across folds. No consistent trend was observed during development regarding optimal architecture configurations with either 80 or 128-dimensional acoustic feature vectors being selected, and depths (number of convolutional blocks) of 5 or 3 with 8 or 16 base channels.

Table 7.5: CNN test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2, 3]	0.4741 ± 0.2983	0.7417 ± 0.0988	$\mathbf{0.3694 \pm 0.0275}$	$\mathbf{0.7202 \pm 0.0405}$
CNN (base)	$\mathbf{0.7556 \pm 0.1626}$	0.4486 ± 0.3389	0.4972 ± 0.1217	0.5927 ± 0.1226
CNN (pt)	0.3222 ± 0.2726	$\mathbf{0.958 \pm 0.05893}$	0.3931 ± 0.0631	0.7102 ± 0.0749

7.2.4. Transformer

We present the test set performance of the Transformer based architecture in Table 7.6. We notice an improvement upon the baseline in sensitivity, EER and AUC (albeit that the latter was marginal). Improvements in EER and AUC are furthered when pre-training, notably AUC, which saw a relative increase of 8.07% with respect to the base system. Moreover, reductions in the standard deviations of the performance metrics are observed, indicating better generalisation. However, sub-optimal decision thresholds degrade the sensitivity of the pre-trained model. We also note that the pre-trained transformer achieves smaller standard deviations for most metrics when compared to its base counterpart - indicating better generalisation across folds. There was no consistent architectural configuration across folds, aside from the size of the embeddings, for which the smallest investigated (128) was optimal throughout.

Table 7.6: Transformers test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2, 3]	0.4741 ± 0.2983	0.7417 ± 0.0988	0.3694 ± 0.0275	0.7202 ± 0.0405
Transformer (base)	$\mathbf{0.7185 \pm 0.1235}$	0.6083 ± 0.0117	0.3278 ± 0.0613	0.7206 ± 0.0696
Transformer (pt)	0.4629 ± 0.1833	$\mathbf{0.8722 \pm 0.1021}$	$\mathbf{0.2819 \pm 0.0585}$	$\mathbf{0.78295 \pm 0.0289}$

Table 7.7: ResNet test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2,3]	0.4741 ± 0.2983	0.7417 ± 0.0988	0.3694 ± 0.0275	0.7202 ± 0.0405
ResNet-18 (base)	0.6815 ± 0.0733	0.3861 ± 0.2756	0.3694 ± 0.0817	0.6495 ± 0.1170
ResNet-50 (base)	$\mathbf{0.7407 \pm 0.2283}$	0.4028 ± 0.3350	0.4833 ± 0.2732	0.6325 ± 0.2628
ResNet-18 (pt)	0.1000 ± 0.1414	$\mathbf{0.9556 \pm 0.0628}$	0.3486 ± 0.0373	$\mathbf{0.7330 \pm 0.0186}$
ResNet-50 (pt)	0.0704 ± 0.0499	0.9347 ± 0.0545	$\mathbf{0.3278 \pm 0.0614}$	0.6714 ± 0.0608

7.2.5. ResNet

We present results for both the ResNet-18 and ResNet-50 architectures in Table 7.7. With the exception of sensitivity, both ResNets perform substantially worse on average across the folds than the LR baseline. However, after pre-training, fair improvements in EER and AUC for both ResNet-50 and 18 are observed. A large reduction in standard deviation is also observed, indicating that pre-training helped to improve generalisation across folds compared to the base models. There was no clear optimal configuration for the ResNets, with both unfreezing the entire backbone, or just the last convolutional block being selected as optimal for different folds with either 80 and 128 -dimensional mel-spectrograms.

7.2.6. AST

The test set performance for the AST architecture is presented in Table 7.8. Although AST is already pre-trained for KWS and hence should be initialised for acoustic tasks, considerably worse performance is observed compared to the baseline with large standard deviations across all metrics. Performance is improved after pre-training, with both EER and AUC surpassing the baseline, as has been generally observed in other architectures.

Table 7.8: AST test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2,3]	$\mathbf{0.4741 \pm 0.2983}$	0.7417 ± 0.0988	0.3694 ± 0.0275	0.7202 ± 0.0405
AST (base)	0.3333 ± 0.1999	$\mathbf{0.9567 \pm 0.0305}$	0.3750 ± 0.1947	0.6984 ± 0.0617
AST (pt)	0.3852 ± 0.2039	0.9361 ± 0.0511	$\mathbf{0.3306 \pm 0.1479}$	$\mathbf{0.7407 \pm 0.0333}$

7.2.7. wav2vec2.0

Test set performance for wav2vec2.0 is presented in Table 7.9. We observe all metrics improve upon the baseline. Interestingly, in contradiction to results observed for other architectures, mean performance for all metrics worsens after pre-training. Interestingly, we do not note the same trend with other architectures, whereby pre-training resulted in sub-optimal decision thresholds and the subsequent large decrease in sensitivity. Rather, we observe a small decrease (5%) similar to that observed for all other metrics. Whilst not all folds benefited from the inclusion of the contextual representation decoder, all optimal configurations chose to unfreeze all layers (including wav2vec2.0's feature extractor) for both the pre-trained and base architectures.

Table 7.9: wav2vec2.0 test set performance, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve. Base and pre-trained models are indicated with "base" and "pt" respectively.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2,3]	0.4741 ± 0.2983	0.7417 ± 0.0988	0.3694 ± 0.0275	0.7202 ± 0.0405
wav2vec2.0	$\mathbf{0.6407 \pm 0.2306}$	$\mathbf{0.8097 \pm 0.1748}$	$\mathbf{0.2791 \pm 0.1244}$	$\mathbf{0.7604 \pm 0.0369}$
wav2vec2.0 (pt)	0.6778 ± 0.2726	0.7639 ± 0.1019	0.3472 ± 0.0579	0.6847 ± 0.0112

7.3. Discussion

To aid in the discussion of the performance of the investigated classifiers, we present a summary of the previously highlighted results in Table 7.10. Our experiments consistently show that the pre-training of networks leads to improved ability in distinguishing between TB and $\overline{\text{TB}}$ (in terms of improved EER and AUC). This is made clear when comparing the performance of corresponding base and pre-trained models which, with the exception of one architecture (wav2vec2.0), always resulted in an improvement in the aforementioned metrics. Concretely, the mean relative percentage improvement in EER and AUC after pre-training computed across all architectures was 18.63% and 5.95% respectively. Conversely, the decision thresholds selected for networks intialised with pre-trained weights were consistently a poorer choice as opposed to the base models across outer folds, evident in the stark reduction in sensitivity and increase in specificity. Despite this shortcoming, pre-training generally leads to a large reduction in the standard deviation of the performance metrics across folds, in particular for AUC and EER, indicating it can improve model generalisation across a diverse set of patients. This is especially important when considering the application of these models as medical triage tools, where a dependable and reliable test is sometimes more important than top-line performance.

We observe that architectures based on sequence processing and aggregating a global temporal context (BiLSTM, Transformer, and wav2vec2.0), generally perform better than the primarily convolution-based or vision-based architectures (CNN, ResNet-18, ResNet-50 and AST), which only focus on capturing local information, albeit at a coarse resolution at later stages in the network. Noting this discrepancy in performance we can state that

Table 7.10: Summary of test set performance across all architectures, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2,3]	0.4741 ± 0.2983	0.7417 ± 0.0988	0.3694 ± 0.0275	0.7202 ± 0.0405
BiLSTM	0.6778 ± 0.0157	0.6278 ± 0.0906	0.3264 ± 0.0098	0.7497 ± 0.0815
BiLSTM-Att	0.6481 ± 0.1142	0.6069 ± 0.0631	0.3708 ± 0.0683	0.7216 ± 0.0759
CNN	$\mathbf{0.7556 \pm 0.1626}$	0.4486 ± 0.3389	0.4972 ± 0.1217	0.5927 ± 0.1226
Transformer	0.7185 ± 0.1235	0.6083 ± 0.0117	0.3278 ± 0.0613	0.7206 ± 0.0696
ResNet-18	0.6815 ± 0.0733	0.3861 ± 0.2756	0.3694 ± 0.0817	0.6495 ± 0.1170
ResNet-50	0.7407 ± 0.2283	0.4028 ± 0.3350	0.4833 ± 0.2732	0.6325 ± 0.2628
AST	0.3333 ± 0.1999	$\mathbf{0.9567 \pm 0.0305}$	0.3750 ± 0.1947	0.6984 ± 0.0617
wav2vec2.0	0.6407 ± 0.2306	0.8097 ± 0.1748	$\mathbf{0.2791 \pm 0.1244}$	$\mathbf{0.7604 \pm 0.0369}$
pre-trained				
BiLSTM	0.3556 ± 0.1841	$\mathbf{0.9583 \pm 0.0589}$	$\mathbf{0.1305 \pm 0.0039}$	$\mathbf{0.7874 \pm 0.0268}$
BiLSTM-Att	0.3185 ± 0.0733	0.9139 ± 0.0610	0.2638 ± 0.0982	0.7387 ± 0.0378
CNN	0.3222 ± 0.2726	0.9580 ± 0.0589	0.3931 ± 0.0631	0.7102 ± 0.0749
Transformer	0.4629 ± 0.1833	0.8722 ± 0.1021	0.2819 ± 0.0585	0.7830 ± 0.0289
ResNet-18	0.1000 ± 0.1414	0.9556 ± 0.0628	0.3486 ± 0.0373	0.7330 ± 0.0186
ResNet-50	0.0704 ± 0.0499	0.9347 ± 0.0545	0.3278 ± 0.0614	0.6714 ± 0.0608
AST	0.3852 ± 0.2039	0.9361 ± 0.0511	0.3306 ± 0.1479	0.7407 ± 0.0333
wav2vec2.0	$\mathbf{0.6778 \pm 0.2726}$	0.7639 ± 0.1019	0.3472 ± 0.0579	0.6847 ± 0.0112

the patterns in the TB signal being relied on for classification are temporal in nature, and require context gathered over an entire cough. This is in contrast to the baseline system, which simply pools acoustic feature vectors over time. Consequently, the architectures that perform the best are BiLTSM, transformer and wav2vec2.0. We present the respective ROC curves in Figure 7.1 for the base (left) and pre-trained variants (right). Without pre-training, the selected architectures appear to match or only slightly improve upon the baseline's initial true positive rates as the decision threshold is decreased and false positives increase. An anecdotal observation indicates that this is improved when these networks are pre-trained, where increases in the initial true positive rates become more clear and more sustained relative to the baseline as the decision threshold is made less strict. Referring to Table 7.10, the BiLSTM improves on the baseline system on practically all metrics (with the exception of specificity) and achieves an EER and AUC of 0.1305 and 0.7874 after pre-training with comparatively low standard deviations of 0.0039 and 0.0268 respectively. This equates to a substantial relative improvement of 9.33% in AUC and 64.67% in EER. Whilst the transformer architecture does not improve upon the baseline initially, after pre-training, we observe a comparable AUC to the BiLSTM, and a decrease in EER by $\approx 14\%$ with reduced standard deviations across folds. Interestingly, whilst wav2vec2.0 achieves the highest AUC without pre-training, it is the only architecture to

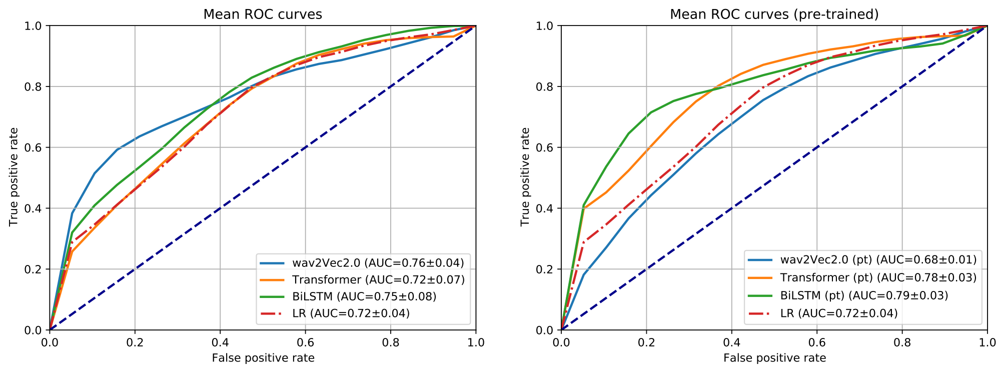


Figure 7.1: Mean ROC curves across the outer folds for three selected architectures (wav2vec2.0, Transformer, and BiLSTM) and the LR baseline for both base (left) and pre-trained variants (right).

degrade in performance afterwards. It is unclear why this may be, and warrants further investigation in future work.

7.3.1. Model size

When comparing the performance of deep architectures, it is important to consider their size for a fair comparison. For high-resource applications, there is typically a trade-off between parameter count and resource usage, where an increase of the former results in improved performance at the expense of the latter. In our work, this relation is not the case, whereby over-parameterisation can lead to over-fitting and subsequent degradation in performance. Moreover, as the intended application of these classifiers is to form part of a triage tool which will most likely be in the form of a mobile application, model size is an important factor to consider. We tabulate the estimated number of model parameters and the respective pre-trained architectures mean test AUC in Table 7.11. Quite clearly (with respect to the BiLSTM architecture), strong performance can be achieved with *small* models. However, overall, the relation between model size and performance is somewhat unclear, partly due to the variance in architectures and their inherent impact on performance.

7.3.2. Decision thresholds

For architectures that performed well, we present the mean EER decision thresholds in Table 7.12. Despite being inferior choices overall, due to the aforementioned reduction in sensitivity and specificity, the standard deviation between decision thresholds is reduced substantially for large pre-trained networks, indicating better generalisation across folds. This improvement in decision threshold generalisation however appears to be related to model size, whereby the smallest model (referring to Table 7.11) actually has the worst decision threshold generalisation. Additionally, it would seem that pre-training, as in [19], drove the selected decision thresholds closer to 0.5, indicating a reduction in class bias. Future work should focus on optimal strategies to select this decision threshold, as it is fundamental to the task.

Table 7.11: Investigated architectures parameter count and respective mean test set AUC with pre-training with exception of the LR baseline.

Model	AUC	Parameter count
LR (*baseline*) [2,3]	0.7202	≈ 128
BiLSTM	0.7874	$\approx$ 55k
BiLSTM-Att	0.7387	$\approx$ 75k
CNN	0.7102	$\approx$ 2.4M
Transformer	0.7831	$\approx$ 1.1M
ResNet-18	0.7333	$\approx$ 11M
ResNet-50	0.6714	$\approx$ 24M
AST	0.7407	$\approx$ 87M
wav2vec2.0	0.6847	$\approx$ 95M

Table 7.12: Mean and standard deviation of EER decision thresholds γ across outer folds for selected architectures.

Model	γ
LR (*baseline*) [2,3]	0.2702 ± 0.1207
BiLSTM	0.6070 ± 0.0538
Transformer	0.3013 ± 0.0750
wav2vec2.0	0.5312 ± 0.0571
pre-trained	
BiLSTM	0.5240 ± 0.1192
Transformer	0.3580 ± 0.0465
wav2vec2.0	0.5160 ± 0.0204

7.3.3. Oracle decision thresholds

As previously highlighted, pre-trained architectures generally improved upon metrics that did not require choosing a decision threshold (EER and AUC). However, the sensitivity and specificity always degraded substantially. Both of these metrics are important if such a classifier should ever be used as a TB triage tool, whereby diagnostic tests have to meet the minimum sensitivity and specificity requirements set by the WHO [85]. As such we investigate the oracle top-line sensitivity and specificity by finding the decision thresholds that would result in achieving the WHO's sensitivity (0.9) and specificity (0.7) minimum requirements respectively. These results are presented in Table 7.13 for the baseline system,

Table 7.13: Oracle specificity and sensitivity of selected classifiers when tuning decision thresholds to meet the WHO's specificity and sensitivity requirements (0.7 and 0.9 respectively).

Model	Sens (Spec≈0.7)	γ_{sp}	Spec (Sens≈0.9)	γ_{se}
LR (*baseline*) [2, 3]	0.5296 ± 0.0052	0.5122 ± 0.0393	0.4750 ± 0.1192	0.2876 ± 0.0272
BiLSTM	0.6074 ± 0.0457	0.7389 ± 0.0715	0.4958 ± 0.1738	0.5341 ± 0.1389
Transformer	0.5370 ± 0.1833	0.5056 ± 0.0793	0.5625 ± 0.0924	0.3101 ± 0.0589
wav2vec2.0	0.6741 ± 0.1440	0.5618 ± 0.0992	0.4764 ± 0.2000	0.4167 ± 0.1763
pre-trained				
BiLSTM	0.7519 ± 0.0367	0.4159 ± 0.1019	0.4750 ± 0.1192	0.2452 ± 0.1263
Transformer	0.7185 ± 0.1235	0.3417 ± 0.0639	0.6083 ± 0.1637	0.2188 ± 0.0542
wav2vec2.0	0.5333 ± 0.1440	0.6195 ± 0.0716	0.4305 ± 0.1383	0.6977 ± 0.2479

and our three best architectures: BiLSTM, transformer, and wav2vec2.0. No classifiers achieve the minimum specifications set by the WHO when using oracle decision thresholds. It should be clarified that when conducting this experiment, decision thresholds were selected that achieved performance specifications either equal to or above and as close to those specified. Due to the relatively small size of each outer folds test set, we were often not able to achieve the target sensitivity and specificity, with the target values sometimes being exceeded by up to 0.05 for each, making our oracle results less accurate. Despite this, some classifiers achieve fair oracle performance. Notably, the pre-trained BiLSTM and transformer architectures achieve a sensitivity of 0.7519 and 0.7185 respectively when fixing specificity to 0.7, which is a substantial improvement over results presented with default decision thresholds. An interesting observation is made when comparing the decision thresholds determined through experimentation presented in Table 7.12 and the oracle decision thresholds used to achieve specificities of 0.7. There appears to be, notably for the transformer architecture, only a small difference ($\approx 0.01 - 0.1$) between the mean decision threshold determined through EER optimisation and the mean decision threshold used to determine oracle specificity results. This relatively small variation indicates some degree of decision threshold sensitivity. Moreover, as highlighted in the previous section, pre-training reduces EER-determined decision threshold standard deviation by a substantial margin for the larger transformer and wav2vec2.0 architectures. This is not the case for oracle decision thresholds, however, where fairly large standard deviations are observed for each architecture.

7.3.4. Sequential forward search

Recall, SFS performs a greedy feature selection, whereby the optimal feature set is iteratively found by, starting from the empty set, adding the single feature that improved performance the most. The subset that resulted in the best performance, often much

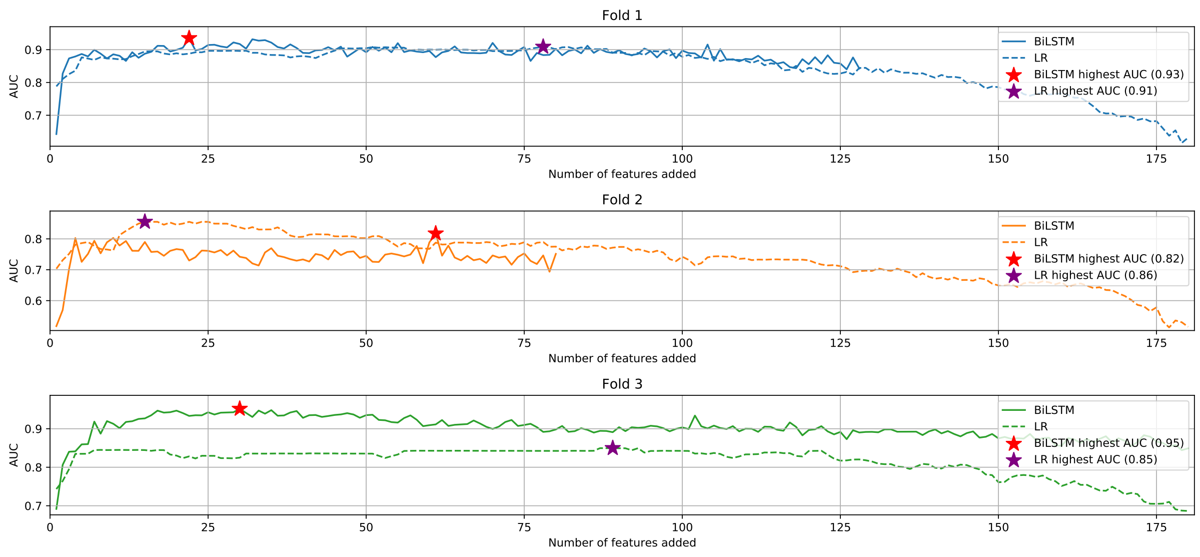


Figure 7.2: Mean development AUCs for each outer fold as the optimal subset of acoustic features is increased during SFS.

smaller than the original feature set size, is chosen. Despite wav2vec2.0 being the best non-pre-trained architecture, we could not use it in SFS experiments. This is due to the fact that it directly receives audio waveforms as input and not acoustic feature vectors. Thus, we choose the next best architecture, the BiLSTM, and compare SFS results to the baseline LR system. We perform SFS for each outer fold independently, with the respective architectural hyper-parameters (model depth, feature size, etc.) found to be optimal for each in previous experimentation. Hence, the optimal feature sizes for each model for each outer fold may not be identical. With this in mind, we present the relation between mean development AUC and the number of features added to the optimal subset by SFS for each outer fold in Figure 7.2. We observe that performance very quickly improves with the use of only a few features and after reaching some maxima begins to slowly decrease. In fact, all architectures need fewer than 10 features to achieve mean development AUCs above 0.8, indicating that there are a few particular frequency bands that are important for classification. The composition of these and whether or not these frequencies generalise across folds will be explored in Chapter 9. Clearly, whilst true for both architectures, LR's performance is highly dependent on the feature subset size, and sharply drops off as it approaches the complete set, indicating the inclusion of all the acoustic features may actually make it harder to learn the TB signal in cough.

We present the mean and the standard deviation of the test set performance across all outer folds for the optimal acoustic feature vector subsets determined for each in Table 7.14. Performance is substantially worse than that expected when we observed Figure 7.2. To this end, it is important to note that during SFS each subsequent additional feature is chosen directly based on its mean development set performance, and thus introduces scope for heavy development set over-fitting. We suspect that this may be the cause for

Table 7.14: Summary of test set performance after performing SFS, mean across outer folds, evaluated through various metrics: sensitivity, specificity, equal error rate, and area under the curve.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
LR (*baseline*) [2,3]	0.1481 ± 0.1386	0.8958 ± 0.1473	0.4569 ± 0.0561	0.6726 ± 0.0367
BiLSTM	0.7519 ± 0.0367	0.6500 ± 0.0707	0.3292 ± 0.1002	0.7374 ± 0.0952

Table 7.15: Comparison of test set performance after performing SFS on the same dataset splits (fold 1) for the BiLSTM architecture for two different experimental code-bases.

Model	Sens ↑	Spec ↑	EER ↓	AUC ↑
BiLSTM [8]	0.6670	0.7500	-	0.8620
BiLSTM *(this study)*	0.7778	0.7500	0.1875	0.8632

the large disparity in performance, although it does not explain why in other work this has not been observed [2,3,8]. Suspecting a mistake in implementation, we compared BiLSTM results for fold 1 to [8], which uses the exact same dataset splits as are represented in fold 1 (albeit an entirely different code-base and optimal architecture configuration). Our results were either similar or improved upon theirs, and to aid in this discussion we present them in Table 7.15. This indicates that the discrepancy in performance is not due to the experiment's implementation, but some other factor.

To investigate whether the discrepancy in development and test performance is due to the hypothesised development set over-fitting, and hence the discovery of a non-generalisable feature subset, we evaluate the BiLSTM architecture on the test set each time a feature was added to the optimal set (that is, for each step in the previously performed SFS). Importantly, these results cannot be used to infer absolute classification performance, but only to investigate the cause for the discrepancy in development and test set performance. Development and test set AUCs for each step in SFS for all three folds are presented in Figure 7.3. Clearly, the feature subset that corresponds to optimal test performance (purple star) requires substantially more features than that during development (red star). This premature peak in development AUC confirms our suspicion that the discrepancy in test set performance as previously presented is due to development set over-fitting, whereby there are more degrees of freedom to do so in the early stages of SFS due to the larger search space.

7.4. Comparison to previous studies

We note that in previous studies [2, 3] stronger classification performance is reported compared to the results presented in this chapter. Notably, both works achieve classification

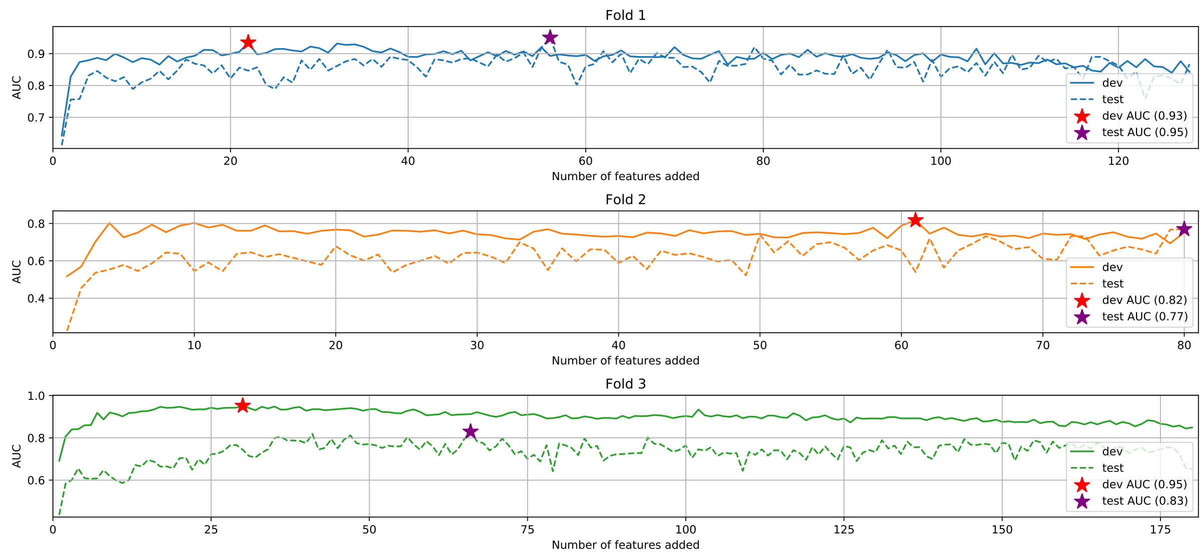


Figure 7.3: Mean BiLSTM test and development AUCs for each fold as the optimal subset of acoustic features is increased during SFS.

performance that meets the WHO standards. By directly comparing the presented performance to these works, it may seem that this study has been unsuccessful in the application of deep architectures to TB cough classification. However, relative to the baseline system, which is in fact the exact architecture used to achieve such performance in the aforementioned studies, our developed architectures make substantial performance improvements. A natural question to ask is why such a discrepancy in performance exists between these works - the dataset size increased, so why did this not result in improved performance as is typically observed in applications of deep learning? In fact, the answer to this question is simple. Despite an increase in dataset size, it was made substantially more complex by spanning two independently collected sets, each comprising its own unique characteristics. The degradation compared to previous studies when tasked with a more complex dataset indicates the task of creating a generalised TB cough classification tool that will perform proficiently in the field may be harder to achieve than initially indicated by exploratory work.

7.5. Summary

In this chapter, we presented classification results regarding the application of various deep learning architectures to TB cough classification. We investigated pre-training these networks on an auxiliary task, namely cough detection. Notably, the pre-trained BiLSTM achieved an EER and AUC of 0.1305 ± 0.0039 and 0.7874 ± 0.0268 respectively, which is a relative improvement over the baseline by 65.68% and 9.33%. We further observed that the pre-training of architectures decreased the standard deviation of performance metrics, whilst typically resulting in improved performance for metrics which did not rely on decision threshold selection. We then detailed observations made with regard to model size, chosen decision thresholds, and an investigation into whether or not chosen architectures could meet the WHO performance standards if oracle decision thresholds could be selected. Lastly, we present classification performance when applying SFS feature selection and reason about the degradation. In the next chapter, we will present a brief investigation into patient identity as a confounding factor in TB cough classification research, and detail experiments conducted in an effort to inhibit the learning of identity by deep neural networks.

Chapter 8

Identity Confounding

In the previous chapter, we noted a degradation in classification performance achieved by our systems (including the baseline system) compared to that reported in previous work [2,3]. Despite the likely hypothesis that this was due to increased dataset complexity, it has been shown that in such small clinical data sets such as the ones used in these studies, machine learning -based methods tend to over-fit to patient identity [86]. When over-fitting to patient identity, large standard deviations in model performance may be observed due to the identity of patients in the test sets mapping (or failing to map) to those seen during training. In this chapter, we qualitatively quantify the extent of potential identity confounding in past work in TB cough classification. This is accomplished through an empirical evaluation whereby a comparison is made to the null hypothesis that models solely rely on patient information to perform TB cough classification, using a similar method to that presented in [86]. We show that while cough does contain TB information, it can easily be overshadowed by models suffering from identity confounding. We then investigate different deep learning techniques to mitigate this issue and present our findings.

8.1. Comparison to the null hypothesis

In initial experimentation preceding the work presented in this study, it was found to be difficult to reproduce the results presented in [3] despite using the same dataset and experimental setup. After a lengthy investigation, it was determined that this was partially due to the use of different nested k-fold splits, that is, the combination of patients appearing in these subsets was different from ours. Upon further investigation, it became apparent that model performance was highly dependent on the composition of these splits. Hence, the exact same experimental setup with different split compositions resulted in substantially different classifier performance. This indicated that some confounding factor was being learnt during training, which only sometimes generalised to the unseen patients.

It has been shown in the literature that surprisingly low speaker identification error rates can be achieved using cough alone [87]. We qualitatively show the extent of identity information contained in cough in Figure 8.1, with a plot of the patient-wise colour-coded t-SNE dimensionality reduction of speaker embeddings from a d-vector extractor trained

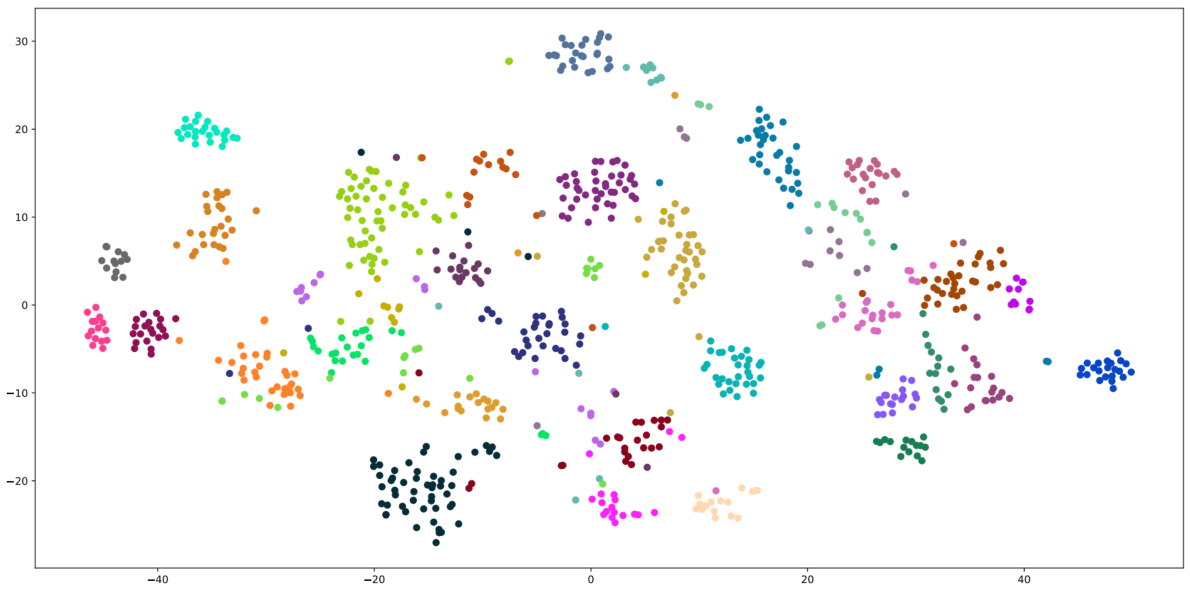


Figure 8.1: Patient-wise colour-coded t-SNE dimensionality reduction of d-vector speaker embeddings extracted using patient coughs.

entirely on speech [88]. The tight clustering of coughs originating from the same patient is observed.

We hypothesise the large deviation in performance when reproducing results in [3] is a result of patient identity confounding and quantify its extent through an empirical evaluation similar to that presented in [86]. In the subsequent experiments, we use the optimal architecture presented in [3] (LR with MFFCs) and the exact experimental setup described in Chapter 2.

8.1.1. Experimental setup

The work presented in [86] provides a framework for a permutation approach to empirically determine the extent of identity confounding in clinical datasets. The premise is to generate a permutation distribution of AUC scores when a classifier's ability to learn any relation between independent (cough) and dependent (TB) variables is inhibited. This simulates the null hypothesis that classifiers do not learn a TB signal, but rather some other confounding factor. This is accomplished by repeatedly assigning random labels (TB or not TB) to all patients in a given dataset. Ideally, there exists a relationship between the information contained in a cough and a patient's TB diagnosis. However, by assigning random TB labels to patients, this relationship is obscured. When a classifier is trained on this obscured data, it is forced to ignore any true relation between the cough feature set and TB - since the true correlation between these variables has been destroyed. Instead, the classifier must learn some other relation in the training data to converge. By conducting a Monte-Carlo simulation, whereby the only variable changed in each iteration is the random seed used to assign new TB labels, and performing nested k-fold cross-validation, the resulting distribution of AUC scores can be estimated and compared to one generated when the relation between the dependent and independent variables is not obscured. This comparison allows the strength of identity confounding to be inferred through qualitative analysis.

We conduct two experiments. The first highlights the extent to which models are relying on patient information during training. This is quantified by comparing distributions observed using the aforementioned permutation approach when patient overlap between train and test sets is allowed (although each patient does not strictly need to appear in all sets due to the number of coughs generated by each, it is unlikely that they do not). Whilst this does not represent the experimental setup used in this work (as we strictly enforce no patient overlap), it is a good way to benchmark the extent to which models rely on identity information to perform TB classification. The second, and more important experiment investigates the overlap between the distribution associated with the null hypothesis and the distribution observed when TB information is intact when no patient overlap is allowed. This is an accurate representation of the experimental setup used in this work. Each Monte-Carlo simulation is described in Table 8.1.

Table 8.1: Monte-Caralo simulations and their respective experimental constraints. Simulation A and C represented the null hypothesis when patient overlap is and is not allowed respectively.

Simulation	**Constraint**
A	• Random labels assigned per patient • Patient overlap
B	• Patient overlap between test and training set
C	• Random labels assigned per patient • No patient overlap between test and training set
D	• No patient overlap between test and training set

8.1.2. Results and discussion

We present the estimated probability densities for AUC and the corresponding mean and variance in Figure 8.2 and Table 8.2 respectively. In agreement with the findings of [86] on voice data, strong identity confounding is present in this dataset. This is indicated by the distribution associated with the null hypothesis being centred at an AUC much larger than 0.5. We observe $f_{AUC|B}(auc|B)$ lies slightly to the right of the null hypothesis distribution, and has lower variance, indicating that the model can learn the disease signal, albeit whilst still fitting to confounding factors. This is reinforced by the stark difference between distributions $f_{AUC|B}(auc|B)$ and $f_{AUC|D}(auc|D)$ whereby the only difference between the experimental setups is patient overlap and as such, over-fitting to confounding factors (such as patient identity) during training substantially impacts performance.

The wide distribution $f_{AUC|D}(auc|D)$ highlights the dependency of model performance on the patients appearing in each split and may also point towards a source for the discrepancy in classifier performance presented in the previous chapter and to that presented in [2, 3] as highlighted in Chapter 2. Although our deep learning experiments were conducted with both the Wallacedene and Brooklyn datasets combined, while previous work was conducted only with the former, the mean of $f_{AUC|D}(auc|D)$ appears to be more representative of the classification performance presented in the previous chapter which is encouraging. However, such a large variance in performance when classifying unseen patients is worrying because it indicates poor generalisation. As it is clear that models over-fit to confounding factors (which could contain patient information) to learn TB labels during training, it is suspected that a partial source for this variance could be the coincidental similar (or dissimilar) patient identification mappings and their respective TB status, and thus dataset splits have a large effect on measurement performance. Despite this, it is quite clear that there is in fact a TB signal that can be learnt in cough, evident when comparing $f_{AUC|D}(auc|D)$ to $f_{AUC|C}(auc|C)$ which on average achieves random classification performance (mean AUC ≈ 0.5). The standard deviation of $f_{AUC|C}(auc|C)$ could have been due to the classifier relying on a host of factors, including random labels that align well with the ground truth.

Table 8.2: Mean and standard deviation of AUC for estimated probability densities generated given conditions A, B, C, and D respectively.

Probability density	AUC
$f_{AUC|A}(auc|A)$	0.8120 ± 0.0395
$f_{AUC|B}(auc|B)$	0.8479 ± 0.0125
$f_{AUC|C}(auc|C)$	0.4993 ± 0.0866
$f_{AUC|D}(auc|D)$	0.6662 ± 0.0448

8.2. Exploring identity-learning mitigation techniques

We previously highlighted that cough contains patient information that is being learnt during training. Moreover, the seeming dependence of model performance on the way the dataset is split into training and testing sets suggests that models are over-fitting to some confounding factor, which could be patient identity. Whilst these experiments were conducted solely on the Wallacedene dataset, there is a possibility that identity confounding may also be present in our combined dataset (Wallacedene + Brooklyn). To explore this, we investigate whether removing patient information has any effect on model

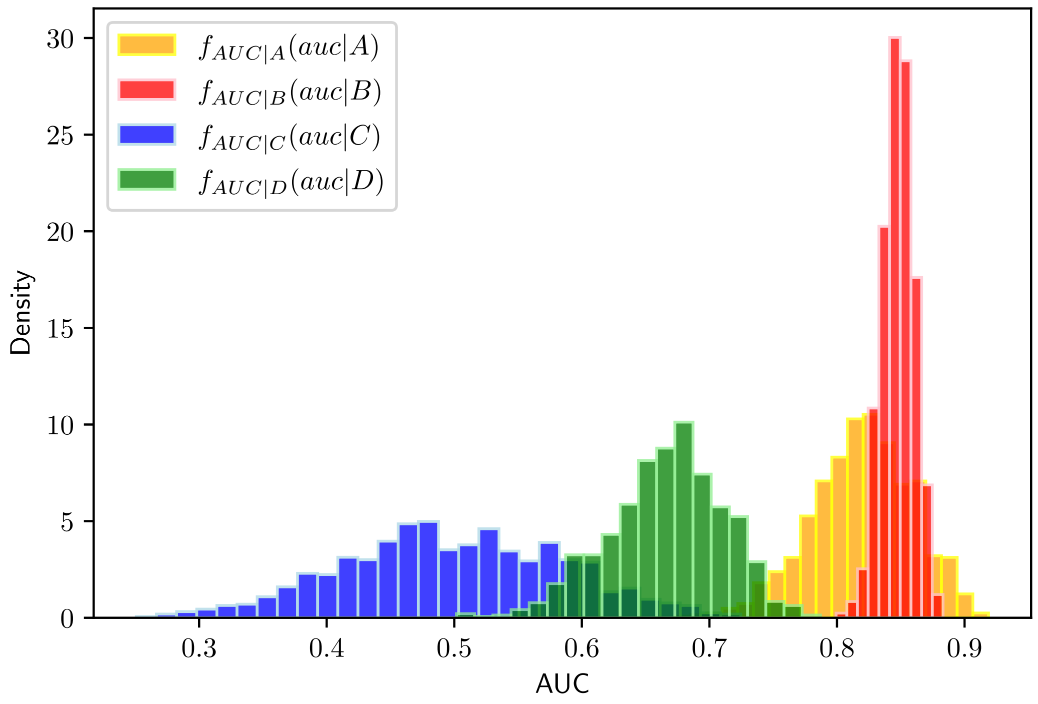


Figure 8.2: Distributions of AUC scores given the experimental conditions described.

performance and more importantly, reduces performance standard deviation when tasked with different dataset set splits.

We experiment with three adapted deep learning techniques to mitigate the learning of patient identity: domain adversarial networks (DANNs), triplet loss, and generalised end-to-end (GE2E) loss. We first describe our experimental setup, including the chosen deep neural architecture with which experimentation will be conducted. We then detail each of the aforementioned mitigation techniques and present experimental results.

8.2.1. Experimental setup

We use the same permutation approach described previously, where a Monte-Carlo simulation is used to generate a distribution of classifier performance, and where the only varying factor between runs is the seed used to perform nested k-fold cross-validation (the same process described in Chapter 6). This process is conducted for a baseline classifier, and the same classifier with each identity-learning inhibitor (DANN, triplet loss and GE2E loss).

Since we are investigating the effect of these mitigation techniques on the performance of deep neural networks, a strong baseline architecture should be used. We choose to use the transformer architecture described in Chapter 6, due to its good performance and fast inference speed. Whilst the BiLSTM performed better and its substantially lower parameter count, its recurrent nature makes it significantly slower and would have resulted in longer run times. Moreover, in this experiment optimising the overall classification performance is not the priority. Instead, we would like to determine its change relative to a baseline system.

To aid the reader, the transformer encoder outputs a single classification token $\boldsymbol{x}_{\text{cls}}$ which is then fed to a linear classification head. We use 128-dimensional mel-spectrograms, sinusoidal positional encodings, 4 transformer encoder blocks with 128-dimensional latent representations, and 4 attention heads. In the identity-learning mitigation techniques described below, the transformer encoder is referred to as the feature extractor, and $\boldsymbol{x}_{\text{cls}}$ is the embedding $\mathbf{e}$.

8.2.2. Domain adversarial training

Domain adversarial training was initially proposed as a representation learning technique that aimed to transfer learnt representations from a source to a target domain by incentivising the network to learn latent representations that are indiscriminate with respect to the differences between the two [89]. A primary neural classifier is trained on the labelled source domain on some supervised task, whilst an auxiliary domain classifier (which shares the same feature extractor as the primary classifier) is tasked with distinguishing between samples originating from the source and from the target domains. Note the target domain is only used to train the auxiliary domain classifier in this task, and not for the primary classification task. The classification loss from both classifiers is back-propagated through the network. However, due to a special gradient reversal layer placed between the feature extractor and auxiliary classifier, which multiplies gradients propagating from the domain classifier by some negative constant, the feature extractor learns domain invariant features. By multiplying the domain classifiers' gradients by this negative constant, the feature extractors' weights are updated in such a way that actually maximises the domain loss (gradient accent). Consequently, when the primary classifier is tasked with the classification of the target domain, it manages to transfer the learnt relations from the source domain well, even though the network never saw the target domain's class labels during training.

More formally, let $G_f(\cdot;\theta_f) = \boldsymbol{e}$ be the neural feature extractor with parameters θ_f , $G_y(\cdot;\theta_y)$ be the primary classifier with parameters θ_y, and $G_d(\cdot;\theta_d)$ be the domain classifier with parameters θ_d. For some input $\mathbf{x}$, let the prediction and domain loss respectively be:

$$\begin{aligned}\mathcal{L}_y(\theta_f,\theta_y) &= \mathcal{L}_y(G_y(G_f(\mathbf{x})),y)\\ \mathcal{L}_d(\theta_f,\theta_d) &= \mathcal{L}_d(G_d(G_f(\mathbf{x})),d)\end{aligned} \tag{8.1}$$

Then the updates for parameters θ_f, θ_y and θ_d using gradient decent where ϵ is the learning rate and λ is the gradient reversal factor are:

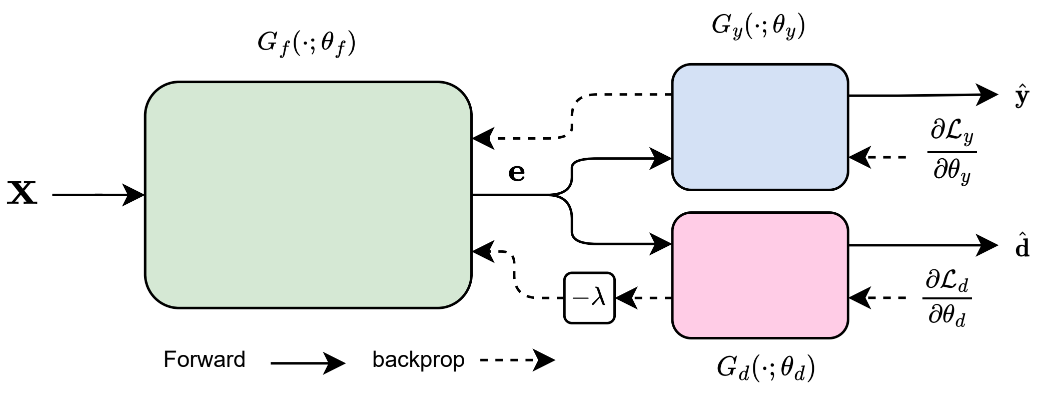


Figure 8.3: A typical domain adversarial neural network. Note the feature extractor (green), the primary classifier (blue) and domain classifier (pink) which is connected to the feature extractor via a gradient reversal layer.

$$\begin{aligned}
\theta_f &\longleftarrow \theta_f - \epsilon\left(\frac{\partial \mathcal{L}_y}{\partial \theta_f} - \lambda \frac{\partial \mathcal{L}_d}{\partial \theta_f}\right), \\
\theta_y &\longleftarrow \theta_y - \epsilon \frac{\partial \mathcal{L}_y}{\partial \theta_y}, \\
\theta_d &\longleftarrow \theta_d - \epsilon\lambda \frac{\partial \mathcal{L}_d}{\partial \theta_d}
\end{aligned} \tag{8.2}$$

The described flow of gradients is visualised in Figure 8.3 for a general DANN. Gradients are only reversed during backpropagation, and the reversal layer simply multiplies the output of $G_f(\cdot)$ by the identity during the forward pass.

We adapt the approach of domain adversarial training by replacing the auxiliary "domain" classifier with a patient classifier, directly forcing the network to learn patient invariant feature representations. In practice, both the primary and auxiliary classifiers are simple FNNs. Following the literature, we make the auxiliary classifier shallower than the primary with two and three hidden layers with ReLU activations respectively, each a quarter the size of the feature extractor's output (the previously described transformer). In our application, we use weighted cross entropy for $\mathcal{L}_y$, and standard cross-entropy for $\mathcal{L}_d$. We exponentially increase λ from 0 at the start of training to 1 at the end [89]. Since there are far more classes to predict for the auxiliary classifier than the primary, the cross-entropy loss is expected to be significantly higher. As such, we weight the domain loss by a constant regularisation term α (either 0.1 or 0.01). Any order of magnitudes higher or lower either destabilised training or simply had no effect. the final loss function is shown below, where B is the batch size, $\boldsymbol{\beta}$ are the class weights for weighted cross-entropy, $\mathbf{y_b}$ and $\hat{\mathbf{y}}_\mathbf{b}$ ground truth and predicted probabilities respectively and $\mathbf{d_b}$ and $\hat{\mathbf{d}}_\mathbf{b}$ are the predicted and true patient identities.

$$\mathcal{L} = -\frac{1}{B}\left(\sum_{b=1}^{B}(\boldsymbol{\beta} \circ \mathbf{y_b} \cdot \log(\hat{\mathbf{y}}_\mathbf{b}) + \alpha\lambda \mathbf{d_b} \cdot \log(\hat{\mathbf{d}}_\mathbf{b}))\right) \tag{8.3}$$

8.2.3. Triplet loss

Originally proposed for image classification [90], triplet loss aims to reduce the distance between embeddings that map to the same class, and maximise the distance between those that do not. This is accomplished by minimising the distance between an anchor embedding $\mathbf{e}$ and another sample of the same class $\mathbf{e}^+$, and maximising the distance between $\mathbf{e}$ and an embedding of another class $\mathbf{e}^-$. This is described in Equation 8.4, where d is a distance function, typically the square Euclidean distance, and m is a margin to incentivise larger distances between embeddings of different classes. This is visualised in Figure 8.4. Observing Equation 8.4, we note that triplet loss allows for distances between embeddings of the same class to be minimised by computing $d(\mathbf{e}, \mathbf{e}^+)$, but also places the same emphasis on maximising $d(\mathbf{e}, \mathbf{e}^-)$. As such, triplet loss allows for some intra-class variance as long as $\mathbf{e}^-$ is sufficiently far away. Owing to this, triplet loss is a good middle ground when investigating the effect of patient identity as a confounding factor, as embeddings produced for different patients of the same TB status do have to map to the same latent region, but not necessarily be identical.

$$L_{\text{triplet}}(\mathbf{e}) = \max(d(\mathbf{e}, \mathbf{e}^+) - d(\mathbf{e}, \mathbf{e}^-) + m, 0) \tag{8.4}$$

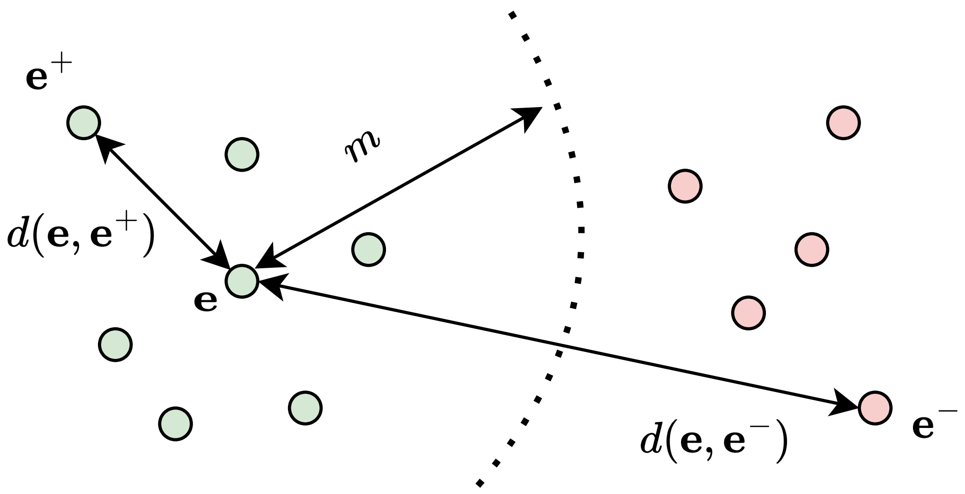


Figure 8.4: Visual representation of the relationship between the anchor $\mathbf{e}$, positive $\mathbf{e}^+$, and negative $\mathbf{e}^-$ pairs for a given triplet. Negative samples are pushed away from the anchor passed the margin, whilst positive samples are brought closer.

In a given batch, there exist a vast number of valid triplet pairs, the selection of which can greatly influence model performance. Simply randomly selecting these pairs may result in a disproportionate number of easy triplets $d(\mathbf{e}, \mathbf{e}^+) \ll d(\mathbf{e}, \mathbf{e}^-)$, as they typically occur more often during training [91] inhibiting the network from learning useful generalised representations. To counter this, a popular approach focuses on hard-mining, whereby the hardest positive (largest $d(\mathbf{e}, \mathbf{e}^{+})$) and negative (smallest $d(\mathbf{e}, \mathbf{e}^{-})$) are selected for a given sample in a randomly sampled mini-batch to form its triplet pair - known as batch hard mining [92]. Whilst with respect to the batch these are hard triplets, globally they are considered *moderate* triplets since they are only determined over a small subset of the dataset. They provide a good compromise between truly hard and easy triplet pairs needed to learn with triplet loss. Formally, batch hard loss for a given embedding $\mathbf{e}$ can be written as below, where P are the total number of positive samples in a batch with respect $\mathbf{e}$, and N is the number of negative samples with respect to $\mathbf{e}$.

$$L_{\mathrm{BH}}(\mathbf{e}) = \max\left(\max_{p=1\ldots P} d(\mathbf{e}, \mathbf{e}_p^{+}) - \min_{n=1\ldots N} d(\mathbf{e}, \mathbf{e}_n^{-}) + m, 0\right) \tag{8.5}$$

We experiment with including an additional triplet loss term to encourage the network to separate embeddings for TB and $\overline{\text{TB}}$ classes whilst reducing the distance between embeddings of the same class albeit with some intra-class variance. The combined loss function used to train our network is given in Equation 8.6, where B is the batch size.

$$\mathcal{L} = -\frac{1}{B}\left(\sum_{b=1}^{B}(\boldsymbol{\beta} \circ \mathbf{y_b} \cdot \log(\hat{\mathbf{y}}_\mathbf{b}) + \alpha \cdot L_{BH}(\mathbf{e}_b))\right) \tag{8.6}$$

The first term in Equation 8.6 is standard weighted cross-entropy. $\mathbf{y_b}$ and $\hat{\mathbf{y}}_\mathbf{b}$ are the vectors of ground truth one-hot labels and predicted probabilities respectively for a given cough in the batch (where the dimension is the number of classes i.e. two), $\boldsymbol{\beta}$ is a vector of class weights. The second is triplet loss, with triplet pairs mined through the batch hard algorithm. To incentivise reducing intra-class variance, we use a lower margin as is typically used in literature (0.3) which places more emphasis on reducing $d(\mathbf{e}, \mathbf{e}_p^{+})$.

8.2.4. GE2E loss

GE2E loss was originally proposed for a speaker diarization task and encourages utterance embeddings to be clustered close together when originating from the same target speaker, whilst maximising the distance between embeddings from different speakers [88]. A similarity matrix $\mathbf{S} \in \mathbb{R}^{N \cdot M \times N}$ is constructed, whose elements are the cosine similarities between the i^{th} embedding vector from the j^{th} speaker $\mathbf{e}_{ji}$ and the speaker centroids $\mathbf{c}_k$ for $1 < j, k < N$ and $1 < i < M$ where N is the number of speakers and M is the (same) number of utterances for each speaker in a batch. This similarity matrix is then used in a loss function, Equation 8.7, to maximise the similarity between $\mathbf{e}_{ji}$ and $\mathbf{c}_j$ (negative term in the equation) whilst minimising the similarity between $\mathbf{e}_{ji}$ and the rest of the speaker centroids.

$$L_{\text{GE2E}}(\mathbf{e}_{ji}) = -S_{ji,j} + \log \sum_{k=1}^{N} \exp(S_{ji,k}) \tag{8.7}$$

In our application, we consider TB and $\overline{\text{TB}}$ coughs to represent two respective "speakers". Hence Equation 8.7 maximises the similarity between the embedding centroids of different patients with the same TB status, whilst minimising the similarity between embeddings from different patients of the same TB status. This encourages the latent space to be patient invariant as it is directly encouraged to be homogeneous for each class. The combined loss function used to train our network is given in Equation 8.8, where B is the batch size.

$$\mathcal{L} = -\frac{1}{B}\left(\sum_{b=1}^{B} \boldsymbol{\beta} \circ \mathbf{y_b} \cdot \log(\hat{\mathbf{y}}_\mathbf{b}) + \alpha \sum_{j,i} L_{GE2E}(\mathbf{e}_{ij})\right) \tag{8.8}$$

Here the first term is standard weighted cross-entropy where $\mathbf{y_b}$ and $\hat{\mathbf{y}}_\mathbf{b}$ are the vectors of ground truth and predicted probabilities respectively (where the dimension is the number of classes i.e. two) for a given cough in the batch and $\boldsymbol{\beta}$ is a vector of class weights. The second term corresponds to the GE2E loss where α is a regularisation parameter and $\mathbf{e}_{ij}$ is the embedding vector of the i^{th} class and j^{th} cough. We consider two options for α, 0.01 and 0.1. Any order of magnitudes higher or lower either destabilised training or generalised to the same solution without the additional loss term.

Table 8.3: Mean and standard deviation of AUC for the baseline architecture and the inclusion of patient-learning mitigation strategy.

Strategy	AUC
Baseline	0.7334 ± 0.0773
DANN	0.7331 ± 0.0786
Triplet loss	0.7238 ± 0.0729
GE2E loss	0.7379 ± 0.0841

8.2.5. Results and discussion

We present the mean and standard deviation for the baseline classifier and each patient-identity learning mitigation strategy in Table 8.3. Importantly, apart from adjusted losses, each training procedure is the same as the baseline. We observe no substantial performance difference between the baseline architecture and each investigated technique. Whilst the inclusion of the GE2E loss term did result in improved mean AUC, the standard deviation increased by 9%. The opposite is observed for triplet loss, where a 6% reduction in standard deviation was observed with a decrease in mean AUC. Lastly, the DANN network neither improved upon the baseline mean nor the standard deviation of AUC.

It is interesting that only triplet loss leads to some decrease in standard deviation in performance across different runs, albeit small. Recall, triplet loss only partially removes patient identity, due to its ability to maintain limited intra-class variance. This indicates that the preservation of some patient identity information is still needed to better generalise across different test split combinations (and the patient compositions thereof). In contrast, the mitigation technique that directly inhibits the network from learning any patient identity information (DANN) performs the worst, further strengthening this idea. Whilst the marginal increase in mean performance when using GE2E loss hints at better generalisation to unseen patients, the increase in standard deviation indicates that its inclusion may have destabilised model convergence, resulting in worse generalisation. A possible source of this destabilisation could have been the fixed run-independent α term, which may have been sub-optimal for the particular fold. Naturally, the aforementioned may be true for each mitigation technique.

Overall, the effect of the investigated patient-learning mitigation strategies appears to be minimal. Whilst strong identity confounding was identified in the Wallacedene dataset, it is possible that due to the almost doubling in the total number of patients in the combined dataset, this was somewhat mitigated. Whilst some techniques did either improve mean performance (GE2E loss) or reduce standard deviation (triplet loss), these improvements were marginal, and no method improved upon both. We note one shortcoming of the experiment was the naive selection of α, which was fixed independently of each fold. It is possible that different training set compositions would have required a unique value for α to better converge, and subsequently impact results.

8.3. Summary

In this chapter, we investigated the extent of identity confounding in data used in previous work, namely the Wallacdene dataset. This was highlighted through an empirical comparison between the distribution associated with the null hypothesis. Specifically, a strong dependence on the patient composition of outer-fold train/test set splits was observed for model performance. We subsequently investigated three deep-learning techniques that could be used to mitigate the learning of patient identity with varying intensity for the dataset used in this work, and compared their inclusion to the performance of a baseline architecture. However, no conclusive improvements were observed. We hypothesised that this may be due to the increase in the number of patients in our study compared to that in previous work, which in turn led to an overall reduction in over-fitting to the identity of patients, mitigating the previously highlighted identity confounding problem. In the next chapter, we briefly explore the acoustic signature of cough for TB classification.

Chapter 9

Acoustic Signature of Cough

In this chapter, we present an investigation into the acoustic signature of cough for TB classification. This is accomplished by selecting networks presented in Chapter 6 and analysing the contents of the feature space that have been learned to be important for classification. This is attempted through three techniques: SFS feature importance analysis, attention weight analysis, and idealised cough representations through adversarial synthesis. Results from this analysis are discussed, and conclusions are made regarding the specific characteristics of TB and $\overline{\text{TB}}$ coughs.

9.1. SFS feature importance

Classification performance when performing SFS on mel-spectrograms was presented in Chapter 7. Despite the choice of a sub-optimal feature set, it was clear that performance improves substantially with the use of only a few acoustic features (or mel-spectrogram bins). Presenting and analysing these specific frequency bins determined to be most important at the beginning of SFS may provide insight to future studies. Observing Figures 7.2 and 7.3, the described sharp increase in performance typically takes place over the first ten discovered optimal features. Although each acoustic feature vector used in each fold has different dimensionality, and therefore different filter bank center frequencies, it may be useful to compare the frequency content being considered as important by each fold to investigate any generalised trends. As such, we present the center frequencies for the first ten discovered optimal features for each fold (sorted to be ascending in frequency) in Figure 9.1. Whilst variance between folds makes it hard to discern reliable conclusions regarding specific important frequency bands, more than half the selected acoustic features have center frequencies below 5000 Hz, indicating that these lower to mid-band frequencies play an important role in carrying the TB signal.

9.2. Temporal analysis

Little is known with regard to which temporal regions of cough are important for TB cough classification. However, this is important if the physiological origin of the learnt

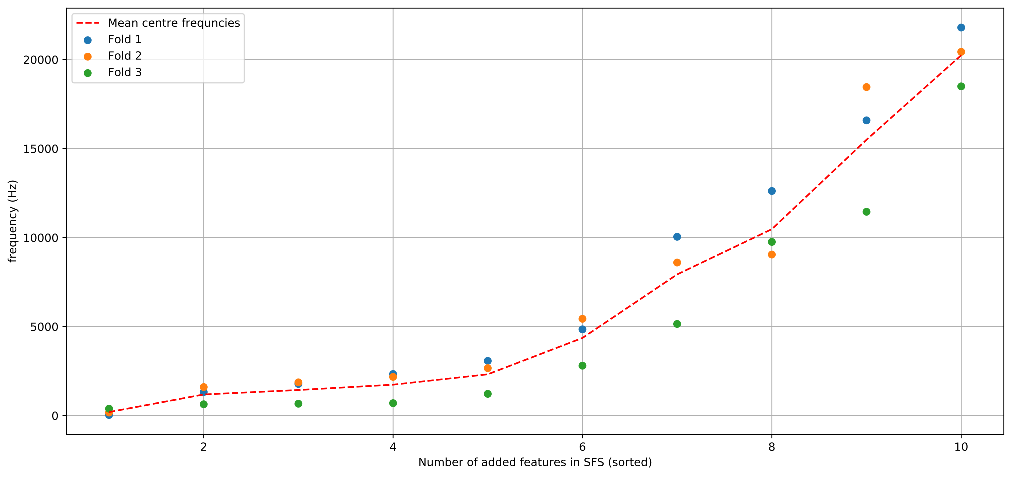


Figure 9.1: Acoustic feature center frequencies for the first ten optimal features discovered by SFS for each fold.

signal is to be determined. For example, it may be that different phases of cough carry information from different regions within the respiratory system (bronchus, trachea and larynx) as highlighted in Chapter 2. Whilst some analysis of important frequencies has been presented in the literature, these were inferred from models trained on the temporal mean of acoustic feature vectors, and as such could give no indication of where this important frequency information occurs in time.

Recall that the BiLSTM-Attention architecture uses an attention mechanism to calculate a weighted sum of the sequence of latent representations outputted by the BiLSTM encoder. Whilst these weightings do not directly relate to a particular time in the input feature space (since the BiLSTM encoder encodes select past and future information in each latent representation), they can be used to infer the general importance of temporal regions relative to one another. We visualised this relation to garner insight into the temporal regions important for cough classification using a simplified BiLSTM-Attention architecture using only one BiLSTM cell with a hidden state size of 32. We present these results in Figure 9.2, where we plot the attention weights as a function of time for six cough mel-spectrograms. The top row corresponds to TB coughs from the same patient, and the bottom row corresponds to $\overline{\text{TB}}$ coughs from originating from a different patient. We observe large importance being placed on regions where the signal has high power and large bandwidth, which coincide with the initial bursts of energy for each coughing episode. Whilst only a few examples are shown, these observations were made in general. Interestingly, this high-energy portion of the coughing sound originates from the lung itself, in particular, the bronchi [1]. It, therefore, appears that whilst TB can manifest in all regions of the respiratory tract, the model is relying on some change in the sound produced inside the lungs of TB and $\overline{\text{TB}}$ patients. Further research is necessary to deduce what the physiological causes of this difference in the audio signals could be.

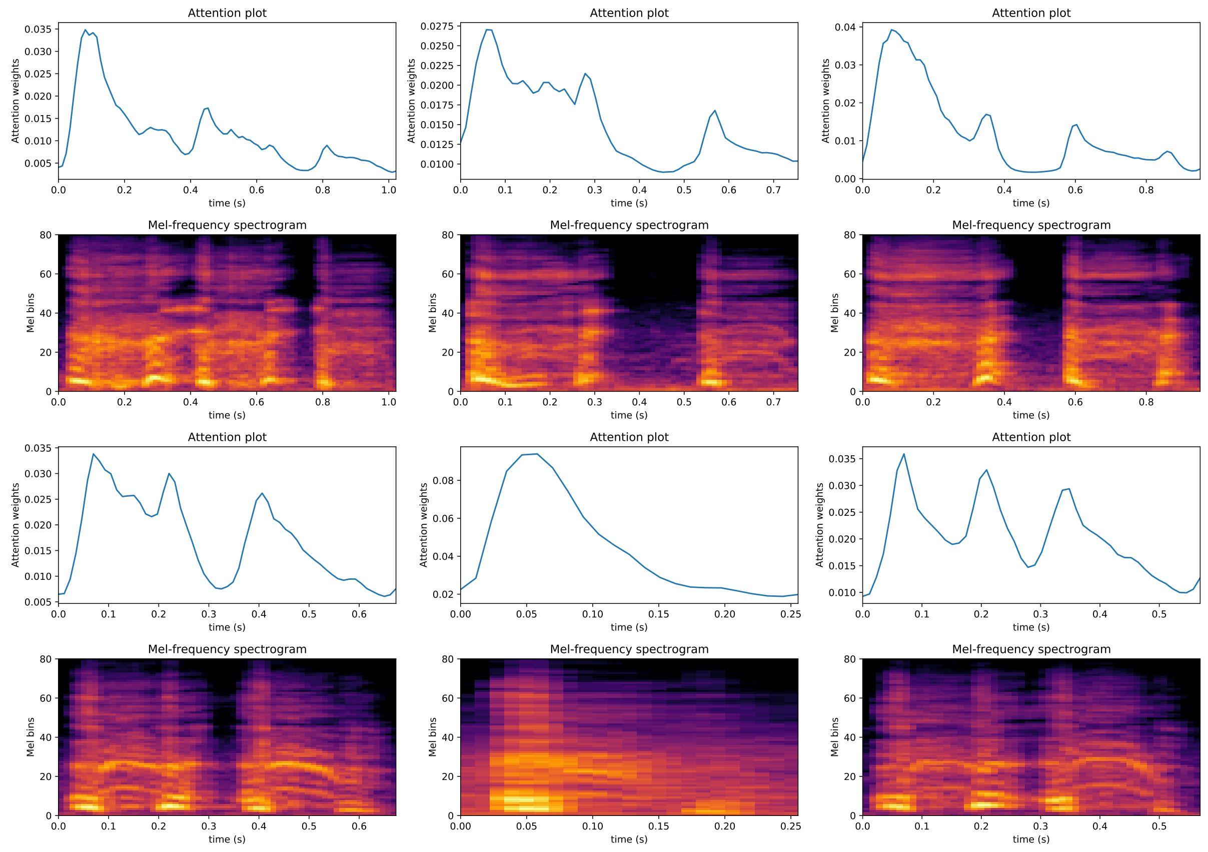


Figure 9.2: Various cough mel-spectrograms and their respective attention weights are shown above them. The first two rows correspond to coughs from the same TB patient, whilst the last two correspond to coughs from a $\overline{\text{TB}}$ patient. Reproduced from [8].

9.3. Idealised coughs

Whilst through attention weight analysis we were able to determine where information learnt to be important lies within the temporal axis of cough and where it is physiologically generated, the distinct characteristics of the signal being learnt remain largely unknown. We conduct an investigation designed to probe the learnt TB and $\overline{\text{TB}}$ representations of trained networks and use this information to infer the distinct characteristics of the signal being learnt. We compare findings from two different networks.

9.3.1. Adversarial synthesis

In an attempt to understand what each network is learning to distinguish between TB and $\overline{\text{TB}}$ coughs, as presented in [8], we synthesise an *idealised* cough for each class by employing a technique that is typically used to generate adversarial attacks. Typically, a trained network maps some input feature vector $\boldsymbol{x}$ to an output $\hat{\boldsymbol{y}}$ using a mapping

$G(\boldsymbol{x};\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ are the learnt parameters of the network. It is possible to estimate the parameterised input $\boldsymbol{x}$ that minimises some loss function $\mathcal{L}$ that is a function of $G(\hat{\boldsymbol{x}};\boldsymbol{\theta})$ where the already trained parameters $\boldsymbol{\theta}$ are fixed and not updated. In doing so, $\mathcal{L}$ is minimised with respect to $\boldsymbol{x}$, resulting in $\hat{\boldsymbol{x}}$. By using cross-entropy for $\mathcal{L}$ as shown in Equation 9.1, and assuming a one-hot encoding for target classes $\boldsymbol{y}$, we can learn the $\hat{\boldsymbol{x}}$ that represents the idealised input for each of these classes by minimising this loss with respect to $\boldsymbol{x}$ (with respect to a trained network).

$$\mathcal{L}_{CE} = \boldsymbol{y} \cdot G(\boldsymbol{x};\boldsymbol{\theta}) \tag{9.1}$$

The flow of gradients during backward propagation is visualised in Figure 9.3.

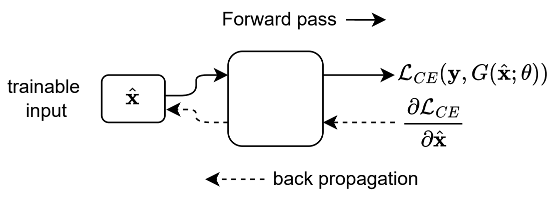

Forward pass
trainable input
$\hat{\mathbf{x}}$
$\mathcal{L}_{CE}(\mathbf{y}, G(\hat{\mathbf{x}};\theta))$
$\frac{\partial \mathcal{L}_{CE}}{\partial \hat{\mathbf{x}}}$
back propagation

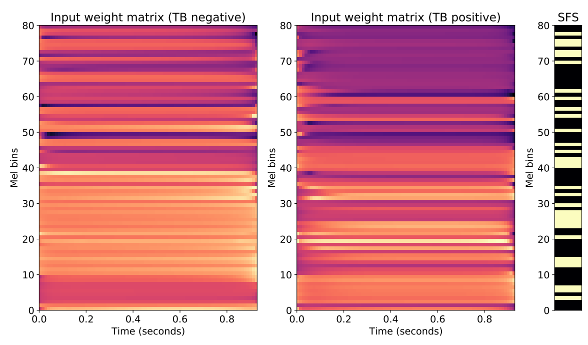
Figure 9.3: Illustration of the loss back-propagation during adversarial synthesis training.

9.3.2. Experimental setup

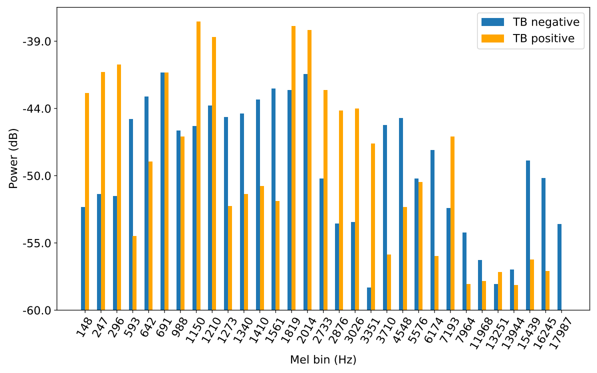
We synthesise idealised inputs for two networks, namely, the BiLSTM-Attention architecture, and the wav2vec2.0 model. The former was chosen to facilitate consistency with the already conducted temporal analysis, and the latter affords the ability to directly synthesise audible waveforms (recall wav2vec2.0 directly takes audio waveforms as input). In the case of the BiLSTM-Attention architecture, we define $\hat{\boldsymbol{x}}$ as the trainable 2D parameter matrix $\hat{\boldsymbol{x}} \in \mathbb{R}^{n \times d_x}$ initialised to zeros, where d_x is the size of the acoustic feature vector seen by the network during training, and n is the number of frames. In this case we select $n = 80$ as it is the aproximate number of frames required for a one-second long audio signal. The same is performed for wav2vec2.0, with $\hat{\boldsymbol{x}} \in \mathbb{R}^s$, where $s = 16,000$ represents a one-second long audio signal.

Next, we perform training as before, but instead of fixing the network input and output and optimising the weights, we fix the network weights and train the input parameter matrices using the previously described adversarial synthesis method. For both networks, we use the Adam optimizer with a learning rate of 1×10^{-3}, and train for as many steps as needed until the training loss converges, and the ideal input has been reached.

We selected the model corresponding to the best development fold from the first outer fold to perform this investigation. When these experiments were originally conducted in [8], this was the singular dataset split being used. In the experimental setup described in Chapter 6, this corresponds to outer fold 1 and inner fold 3. We select the optimal



(a) Idealised mel-spectrograms for TB negative (left) and positive (center) coughs, with bins identified by SFS shown in yellow (right).



(b) Mean power vs center frequency of features deemed most important by SFS of for each idealised mel-spectrogram.

Figure 9.4: Idealised mel-spectrograms and mean spectral power for the synthesised coughs produced by the BiLSTM-Attention architecture. Reproduced from [8].

wav2vec2.0 model that was trained on this fold such that we can conduct a fair comparison between each network's idealised inputs. Importantly, the BiLSTM-Attention architecture was trained on mel-spectrograms extracted from 44.1kHz audio, whilst wav2vec2.0 was trained with cough audio down-sampled to 16kHz, and thus restricts the comparison of frequencies learnt to be important to 8kHz (the Nyquist theorem).

9.3.3. Results and discussion

Figure 9.4a depicts the idealised coughs synthesised by the BiLSTM-Attention network using the adversarial synthesis method described previously. Clear differences between idealised $\overline{\text{TB}}$ and TB cough mel-spectrograms are observed. This is further illustrated by comparing the mean power of these idealised coughs as a function of the frequencies determined to be most important by SFS, as shown in Figure 9.4b. Note this was a separate SFS experiment from that presented in Chapter 7. For the idealised TB cough, we observe generally higher power at lower frequencies (< 500Hz) and the mid-band range of 1.8kHz $-$ 3.3kHz whereas the $\overline{\text{TB}}$ cough has higher power between 1.2kHz $-$ 1.8kHz and frequencies greater than 3.7kHz, which include frequencies far outside the typical range of human speech (> 8kHz).

Next we present the idealised waveforms[1] produced by the wav2vec2.0 architecture as shown in Figure 9.5. Initial inspection yields very little discernible difference between the two synthesised coughs, with both waveforms largely resembling noise, even upon auditory inspection. However, when computing the mel-spectrogram of each as presented in Figure 9.6a, clear differences emerge. As with the BiLSTM-Attention architecture, the idealised $\overline{\text{TB}}$ cough tends to have a wide band of higher power, whereas the idealised TB coughs power is generally localised in low frequencies. This is reaffirmed when plotting

[1]Recordings are made available at: https://youtu.be/r3AvD5mE8Fw ($\overline{\text{TB}}$) and https://youtu.be/ow5JoAJE5zY (TB).

the mean spectral power for the center frequency mel-bin (on a log scale) in Figure 9.6b, whereby we observe higher power for the TB cough for frequencies below 500Hz, whilst the converse is true until around 1.5kHz, where an interesting phenomenon is observed. There appears to be a narrow band of high power centered at 1.5kHz which both idealised coughs contain, the origin of which is unknown. Greater periodicity is observed in the $\overline{\text{TB}}$ coughs mean powers, whereby the power oscillates from about 2.5kHz, with a $\approx$ 1kHz period. The same phenomenon is not observed as predominantly for the TB cough, for which the power slowly dissipates as the frequency increases. These same oscillations are observed for results presented in Figure 9.4b, albeit presented at a different scale.

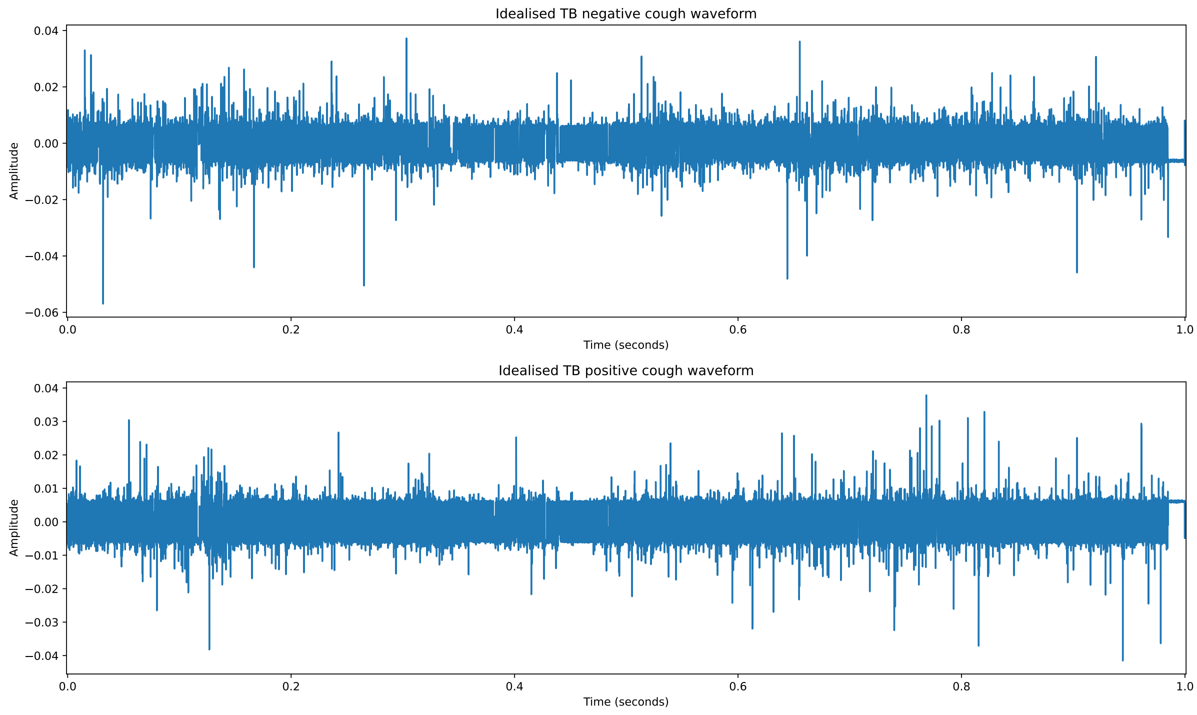
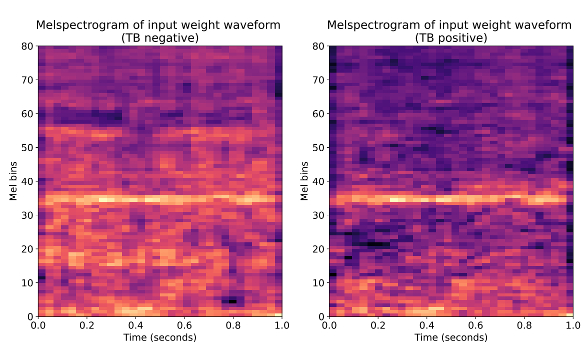
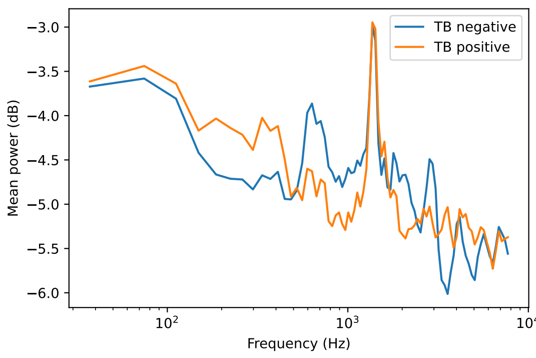


Figure 9.5: Idealised waveforms for a TB negative (top) and TB positive (bottom) cough produced by the wav2vec2.0 architecture.



(a) Mel-spectrograms of idealised cough waveforms for TB negative (left) and positive (right) coughs.



(b) Mean power of each idealised mel-spectrogram vs mel bin center frequency on log scale.

Figure 9.6: Mel-spectrograms of and mean spectral power of synthesised cough waveforms produced by the wav2vec2.0 architecture.

9.4. Summary

In this chapter, we presented a brief investigation into the characteristics learnt to be important for selected deep architectures (BiLSTM, BiLSTM-Attention and wav2vec2.0). We presented a brief exploration of the particular features determined to be important by the SFS search performed in Chapter 7. Utilising the attention-based architecture, the importance of certain temporal regions in the cough signal could be visualised. It was observed that the initial voiced regions of cough were the most important for classification, providing evidence that the TB signal being learnt does indeed originate in the lungs. Moreover, by employing a neural adversarial synthesis technique, idealised TB negative and positive coughs were formulated. Subsequent inspection revealed stark differences between the energy content in specific frequency bands. This was then repeated for the wav2vec2.0 architecture, whereby audible coughs were synthesised and further analysed. The two idealised representations generated by each architecture were compared, and a large degree of overlap between the distinct characteristics of the learnt signal was observed, providing new insights into the aspects of a tuberculosis cough that are important for classification. In the next and final chapter, we present a final summary of the work conducted in this study.

Chapter 10

Conclusion

10.1. Summary and conclusions

This thesis has explored the application of deep learning -based methods for TB cough classification. This was motivated by a lack of such techniques being thoroughly explored in the literature (or the success thereof), which has mostly relied on simple machine learning algorithms such as logistic regression. We began by reviewing literature relevant to this study, namely work completed in cough analysis and cough classification for several respiratory ailments. As cough classification is a highly under-researched field, we also briefly explored notable work in other acoustic analysis domains which focus on deep learning -based solutions, including acoustic scene classification, keyword spotting, and automatic speech recognition. We subsequently detailed the fundamental background of deep learning building blocks and architectures relevant to this work. We also provided detailed descriptions and motivation for the use of various pre-trained architectures to be fine-tuned for cough classification (ResNet, AST, wav2vec2.0). Next, we described the datasets used in our experiments, including how datasets from previous work were combined to present a larger, more diverse, and challenging set. In addition, a dataset constructed for a cough detection pre-training task is also described.

Once sufficient context was established, we began detailing the setup of the experiments conducted to investigate the application of deep learning -based methods for TB cough classification. This included the structure of the proposed architectures (BiLSTM, BiLSTM-Attention, CNN, transformer, ResNet, AST and wav2vec2.0), how these networks were trained, acoustic feature vectors, nested cross-validation, and evaluation metrics. Results regarding an initial investigation into data augmentation techniques and optimal acoustic feature representations with select networks were presented, with speed-perturbation being determined to be a useful augmentation technique, whilst mel-spectrograms performed best among investigated acoustic representations. Next, the classification performance of each investigated network was presented with and without pre-training and compared to a baseline logistic regression system. Optimal configurations determined across each independent test fold for each architecture were briefly discussed, whereby it became clear that there was a large degree of variation in the chosen best hyper-parameters. Upon inspection of results, architectures that were designed with temporally structured data in mind generally performed better than their CNN counterparts both before and after pre-training and surpassed baseline performance by a substantial margin. This indicated that preserving the temporal structure of cough is imperative for improved classification performance, although this had been disregarded in previous studies. Notably, the pre-trained BiLSTM achieved an EER and AUC of 0.1305 ± 0.0039 and 0.7874 ± 0.0268 respectively, which is a relative improvement over the baseline by 65.68% and 9.33%. We further observed that the pre-training of architectures decreased the standard deviation of performance metrics by a large margin, implying better generalisation. Moreover, pre-training typically resulted in improved performance for all systems, specifically in terms of EER and AUC.

This was followed by a small study into the impact of patient identity as a confounding factor when performing TB cough classification. We showed that cough contains a large degree of identity information, which classifiers in previous work learnt during training. Based on this observation, we investigated if employing techniques to reduce the ability of deep learning-based architectures to learn patient identity improved classification generalisation across patients. This included the application of domain adversarial networks, triplet loss, and GE2E loss. Although triplet loss decreased the standard deviation of classification performance (AUC) across different test set patient compositions, and GE2E loss improved mean performance, none of the investigated techniques improved both. This indicated that even though networks may learn patient identity during training, it does not supersede the generalised TB signal being learnt.

Lastly, we concluded this study with exploratory analysis into distinct characteristics of cough that were being learnt for TB classification. First, we presented some of the most important frequency bands for classification, determined through a sequential forward feature selection search. It became clear that whilst higher frequencies were important, the majority of the frequency bins deemed to be most important were centred at frequencies lower than 5kHz. Next, visualisations of the temporal attention scores produced by the BiLSTM-Attention architecture for numerous cough mel-spectrograms were presented. It was concluded that the network was mostly relying on initial explosive regions of cough for classification, which corresponds to the portion of the coughing sound that typically originates from the bronchi and provided evidence that the characteristic TB signal being learnt originates in the lungs of patients. Finally, an adversarial synthesis technique was employed to deduce idealised mel-spectrograms (using the BiLSTM-Attention architecture) and raw audio waveforms (using wav2vec2.0) for TB and non-TB coughs. Subsequent inspection revealed stark differences between the energy content in specific frequency bands for both and provided new insights into the distinct characteristics of a TB cough that enable its use for classification.

10.2. Future work

In future work, emphasis should be placed on the rigorous testing of TB cough classifiers on larger datasets. This is important if such classifiers are to be implemented in a clinical setting whereby a larger degree of certainty in expected performance is needed. Not only would larger datasets allow for less noisy development owing to the better generalisation between development and test sets, but it would also allow for better model generalisation overall due to the increase in unique patients and recording environments seen during training. Additionally, as such classifiers will eventually be deployed in a mobile application, the development of future classifiers should be performed with this in mind, and techniques such as quantisation and knowledge distillation should be explored.

In this work, some results were negatively impacted by the sub-optimal selection of decision thresholds. This problem arose due to the aggregation of model predictions across outer folds, and the small size of development sets. As the decision threshold-based metrics are imperative for the success of these models should they be deployed in a real-world TB screening tool, future work should optimise the strategy used for decision threshold selection.

Bibliography

[1] J. Korpáš, J. Sadloňová, and M. Vrabec, "Analysis of the cough sound: an overview," *Pulmonary Pharmacology*, vol. 9, no. 5-6, pp. 261–268, 1996.

[2] G. Botha, G. Theron, R. Warren, M. Klopper, K. Dheda, P. Van Helden, and T. Niesler, "Detection of tuberculosis by automatic cough sound analysis," *Physiological Measurement*, vol. 39, no. 4, 2018.

[3] M. Pahar, M. Klopper, B. Reeve, G. Theron, R. Warren, and T. Niesler, "Automatic cough classification for tuberculosis screening in a real-world environment," *Physiological Measurement*, vol. 42, no. 10, 2021.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[5] Y. Gong, Y.-A. Chung, and J. Glass, "AST: Audio Spectrogram Transformer," in *Proceedings of Interspeech*, 2021, pp. 571–575.

[6] D. Seo, H.-S. Oh, and Y. Jung, "wav2kws: Transfer learning from speech representations for keyword spotting," *IEEE Access*, vol. 9, pp. 80 682–80 691, 2021.

[7] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A simple data augmentation method for automatic speech recognition," in *Proceedings of Interspeech*, 2019, pp. 2613–2617.

[8] G. Frost, G. Theron, and T. Niesler, "TB or not TB? Acoustic cough analysis for tuberculosis classification," in *Proceedings of Interspeech*, 2022, pp. 2448–2452.

[9] W. H. Organization *et al.*, "Global tuberculosis report 2021," World Health Organization, Tech. Rep., 2021.

[10] N. Foster, A. Vassall, S. Cleary, L. Cunnama, G. Churchyard, and E. Sinanovic, "The economic burden of TB diagnosis and treatment in South Africa," *Social Science and Medicine*, vol. 130, pp. 42–50, 2015.

[11] A. Roguin, "Rene theophile hyacinthe laënnec (1781–1826): the man behind the stethoscope," *Clinical Medicine and Research*, vol. 4, no. 3, pp. 230–235, 2006.

[12] Y. A. Amrulloh, D. A. Wati, F. Pratiwi, and R. Triasih, "A novel method for wet/dry cough classification in pediatric population," in *Proceedings of the IEEE Region 10 Symposium (TENSYMP)*, 2016, pp. 125–129.

[13] V. Swarnkar, U. R. Abeyratne, A. B. Chang, Y. A. Amrulloh, A. Setyati, and R. Triasih, "Automatic identification of wet and dry cough in pediatric patients with respiratory diseases," *Annals of Biomedical Engineering*, vol. 41, no. 5, pp. 1016–1028, 2013.

[14] V. Swarnkar, U. R. Abeyratne, Y. A. Amrulloh, and A. Chang, "Automated algorithm for wet/dry cough sounds classification," in *Proceedings of the IEEE conference of the Engineering in Medicine and Biology Society (EMBC)*, 2012, pp. 3147–3150.

[15] Y. Amrulloh, U. Abeyratne, V. Swarnkar, and R. Triasih, "Cough sound analysis for pneumonia and asthma classification in pediatric population," in *Proccedings of the IEEE International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, 2015, pp. 127–131.

[16] U. R. Abeyratne, V. Swarnkar, A. Setyati, and R. Triasih, "Cough sound analysis can rapidly diagnose childhood pneumonia," *Annals of Biomedical Engineering*, vol. 41, no. 11, pp. 2448–2462, 2013.

[17] 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*, 2021, epub 2021 June 17.

[18] P. Mouawad, T. Dubnov, and S. Dubnov, "Robust detection of COVID-19 in cough sounds," *SN Computer Science*, vol. 2, no. 1, pp. 1–13, 2021.

[19] P. Bagad, A. Dalmia, J. Doshi, A. Nagrani, P. Bhamare, A. Mahale, S. Rane, N. Agarwal, and R. Panicker, "Cough against COVID: Evidence of COVID-19 signature in cough sounds," *arXiv preprint arXiv:2009.08790*, 2020.

[20] B. Agbley, J. Li, A. Haq, B. Cobbinah, D. Kulevome, P. Agbefu, and B. Eleeza, "Wavelet-based cough signal decomposition for multimodal classification," in *Proc-ceedings of the IEEE International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2020, pp. 5–9.

[21] "TB Statistics South Africa," Available at https://tbfacts.org/tb-statistics-south-africa/ (2022/06/02).

[22] R. Long, M. Divangahi, and K. Schwartzman, "Chapter 2: Transmission and pathogenesis of tuberculosis," *Canadian Journal of Respiratory, Critical Care, and Sleep Medicine*, vol. 6, pp. 22–32, 2022.

[23] C. M. Muvunyi, F. Masaisa, and P. Cardona, "Diagnosis of smear-negative pulmonary tuberculosis in low-income countries: current evidence in Sub-Saharan Africa with special focus on HIV infection or AIDS," *Understanding Tuberculosis–Global Experiences and Innovative Approaches to the Diagnosis*, 2012.

[24] S. Nayak and B. Acharjya, "Mantoux test and its interpretation," *Indian Dermatology Online Journal*, vol. 3, no. 1, p. 2, 2012.

[25] World Health Organization *et al.*, "Chest radiography in tuberculosis detection: summary of current WHO recommendations and guidance on programmatic approaches," World Health Organization, Tech. Rep., 2016.

[26] "Genexpert - testing for TB & drug resistant TB," Available at https://tbfacts.org/genexpert/ (2022/06/02).

[27] J. D. Lewis, "Acoustic scanning of human lungs for clinical diagnosis," Ph.D. dissertation, Northwestern University, 1982.

[28] A. Murata, Y. Taniguchi, Y. Hashimoto, Y. Kaneko, Y. Takasaki, and S. Kudoh, "Discrimination of productive and non-productive cough by sound analysis," *Internal Medicine*, vol. 37, no. 9, pp. 732–735, 1998.

[29] H. Chatrzarrin, A. Arcelus, R. Goubran, and F. Knoefel, "Feature extraction for the differentiation of dry and wet cough sounds," in *Proceedings of the IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, 2011, pp. 162–166.

[30] "UNICEF Data - Child Statistics: Pneumonia," Available at https://data.unicef.org/ (2022/10/24).

[31] N. Sharma, P. Krishnan, R. Kumar, S. Ramoji, S. R. Chetupalli, N. R., P. K. Ghosh, and S. Ganapathy, "Coswara — A Database of Breathing, Cough, and Voice Sounds for COVID-19 Diagnosis," in *Proceedings of Interspeech*, 2020, pp. 4811–4815.

[32] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[33] F. J. Ferri, P. Pudil, M. Hatef, and J. Kittler, "Comparative study of techniques for large-scale feature selection," *Machine Intelligence and Pattern Recognition*, vol. 14, pp. 403–413, 1994.

[34] 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, 2022, epub 2021 Dec 17.

[35] K. J. Piczak, “ESC: Dataset for Environmental Sound Classification,” in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015, pp. 1015–1018.

[36] F. Font, G. Roma, and X. Serra, “Freesound technical demo,” in *Proceedings of the 21st ACM international conference on Multimedia*, 2013, pp. 411–412.

[37] F. Al Hossain, A. A. Lover, G. A. Corey, N. G. Reich, and T. Rahman, “Flusense: a contactless syndromic surveillance platform for influenza-like illness in hospital waiting areas,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, vol. 4, no. 1, pp. 1–28, 2020.

[38] L. Orlandic, T. Teijeiro, and D. Atienza, “The COUGHVID crowdsourcing dataset, a corpus for the study of large-scale cough analysis algorithms,” *Scientific Data*, vol. 8, no. 1, pp. 1–10, 2021.

[39] World Health Organization *et al.*, “High-priority target product profiles for new tuberculosis diagnostics: report of a consensus meeting,” World Health Organization, Tech. Rep., 2014.

[40] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, “PANNs: Large-scale pretrained audio neural networks for audio pattern recognition,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 2880–2894, 2020.

[41] Y. Gong, Y.-A. Chung, and J. Glass, “PSLA: Improving audio tagging with pretraining, sampling, labeling, and aggregation,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 3292–3306, 2021.

[42] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[43] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, “Audio set: An ontology and human-labeled dataset for audio events,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 776–780.

[44] P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition,” *arXiv preprint arXiv:1804.03209*, 2018.

[45] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, “A neural attention model for speech command recognition,” *arXiv preprint arXiv:1808.08929*, 2018.

[46] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," *arXiv preprint arXiv:2005.06720*, 2020.

[47] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an ASR corpus based on public domain audio books," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5206–5210.

[48] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the International Conference on Machine learning (ICML)*, 2006, pp. 369–376.

[49] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning.* MIT press Cambridge, 2016, vol. 1, no. 2.

[50] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[52] A. Rakitianskaia and A. Engelbrecht, "Measuring saturation in neural networks," in *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI)*, 2015, pp. 1423–1430.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[54] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[55] L. N. Smith, "Cyclical learning rates for training neural networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2017, pp. 464–472.

[56] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[57] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[58] R. Hibare and A. Vibhute, "Feature extraction techniques in speech processing: A survey," *International Journal of Computer Applications*, vol. 107, no. 5, 2014.

[59] R. Hasan, M. Jamil, G. Rabbani, and S. Rahman, "Speaker identification using mel frequency cepstral coefficients," in *Proceedings of the International Conference on Electrical Computer Engineering (ICECE)*, 2004, pp. 565–568.

[60] Q. Li, Y. Yang, T. Lan, H. Zhu, Q. Wei, F. Qiao, X. Liu, and H. Yang, "MSP-MFCC: Energy-efficient MFCC feature extraction method with mixed-signal processing architecture for wearable speech recognition applications," *IEEE Access*, vol. 20, pp. 1–9, 03 2020.

[61] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques," *Journal of Computing*, vol. 2, pp. 138 – 143, 03 2010.

[62] C. J. Kelly, A. Karthikesalingam, M. Suleyman, G. Corrado, and D. King, "Key challenges for delivering clinical impact with artificial intelligence," *BMC Medicine*, vol. 17, no. 1, pp. 1–9, 2019.

[63] C. Leng, "Discriminating coughs in a multi-bed ward environment," Master's thesis, Stellenbosch University, 2021.

[64] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 449–12 460, 2020.

[65] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[66] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[67] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 3523–3542, 2021.

[68] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[69] I. Bello, W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T.-Y. Lin, J. Shlens, and B. Zoph, "Revisiting resnets: Improved training and scaling strategies," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22 614–22 627, 2021.

[70] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Computing Surveys*, vol. 54, pp. 1–41, 2021.

[71] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2021, pp. 10 347–10 357.

[72] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *arXiv preprint arXiv:2106.04554*, 2021.

[73] S. wen Yang, P.-H. Chi, Y.-S. Chuang, C.-I. J. Lai, K. Lakhotia, Y. Y. Lin, A. T. Liu, J. Shi, X. Chang, G.-T. Lin, T.-H. Huang, W.-C. Tseng, K. tik Lee, D.-R. Liu, Z. Huang, S. Dong, S.-W. Li, S. Watanabe, A. Mohamed, and H. yi Lee, "SUPERB: Speech Processing Universal PERformance Benchmark," in *Proceedings of Interspeech*, 2021, pp. 1194–1198.

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

[75] F. Weninger, H. Erdogan, S. Watanabe, E. Vincent, J. Le Roux, J. Hershey, and B. Schuller, "Speech enhancement with LSTM recurrent neural networks and its application to noise-robust ASR," in *12th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, 2015.

[76] A. Zeyer, P. Doetsch, P. Voigtlaender, R. Schlüter, and H. Ney, "A comprehensive study of deep bidirectional LSTM RNNs for acoustic modeling in speech recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 2462–2466.

[77] J. Billa, "ISI ASR System for the Low Resource Speech Recognition Challenge for Indian Languages," in *Proceedings of Interspeech*, 2018, pp. 3207–3211.

[78] F. Eyben, F. Weninger, S. Squartini, and B. Schuller, "Real-life voice activity detection with LSTM recurrent neural networks and an application to Hollywood movies," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 483–487.

[79] N. Wilkinson and T. Niesler, "A hybrid CNN-BiLSTM voice activity detector," in *Proceedings fo the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 6803–6807.

[80] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[81] J. D. M.-W. C. Kenton and L. K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2019, pp. 4171–4186.

[82] Y. A. Ioannou, "Structural priors in deep neural networks," Ph.D. dissertation, University of Cambridge, 2018.

[83] I. D. Miranda, A. H. Diacon, and T. R. Niesler, "A comparative study of features for acoustic cough detection using deep architectures," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2019, pp. 2601–2605.

[84] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Proceedings of Interspeech*, 2015.

[85] World Health Organization *et al.*, "High priority target product profiles for new tuberculosis diagnostics: Report of a consensus meeting," World Health Organization, Tech. Rep., 2014.

[86] E. C. Neto, A. Pratap, T. M. Perumal, M. Tummalacherla, P. Snyder, B. M. Bot, A. D. Trister, S. H. Friend, L. Mangravite, and L. Omberg, "Detecting the impact of subject characteristics on machine learning-based diagnostic applications," *NPJ Digital Medicine*, vol. 2, no. 1, pp. 1–6, 2019.

[87] M. Zhang, Y. Chen, L. Li, and D. Wang, "Speaker recognition with cough, laugh and "wei"," in *Proceedings of the IEEE Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2017, pp. 497–501.

[88] L. Wan, Q. Wang, A. Papir, and I. L. Moreno, "Generalized end-to-end loss for speaker verification," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4879–4883.

[89] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[90] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *International Workshop on Similarity-based Pattern Recognition*. Springer, 2015, pp. 84–92.

[91] G. Kertész, "Different triplet sampling techniques for lossless triplet loss on metric similarity learning," in *Proceedings of the IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2021, pp. 449–454.

[92] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.

Appendix A

Published work

A.1. Interspeech 2022

Conference paper published in the proceedings of Interspeech, 2022, Incheon, South Korea [8].

TB or not TB? Acoustic cough analysis for tuberculosis classification

Geoffrey Frost[1], *Grant Theron*[2], *Thomas Niesler*[1]

[1]Department of Electrical and Electronic Engineering, Stellenbosch University, South Africa
[2]SAMRC Centre for Tuberculosis Research, DSI/NRF Centre of Excellence for Biomedical Tuberculosis Research, Stellenbosch University, South Africa

{gfrost, gtheron, trn}@sun.ac.za

Abstract

In this work, we explore recurrent neural network architectures for tuberculosis (TB) cough classification. In contrast to previous unsuccessful attempts to implement deep architectures in this domain, we show that a basic bidirectional long short-term memory network (BiLSTM) can achieve improved performance. In addition, we show that by performing greedy feature selection in conjunction with a newly-proposed attention-based architecture that learns patient invariant features, substantially better generalisation can be achieved compared to a baseline and other considered architectures. Furthermore, this attention mechanism allows an inspection of the temporal regions of the audio signal considered to be important for classification to be performed. Finally, we develop a neural style transfer technique to infer idealised inputs which can subsequently be analysed. We find distinct differences between the idealised power spectra of TB and non-TB coughs, which provide clues about the origin of the features in the audio signal.

Index Terms: cough, tuberculosis (TB), BiLSTM, attention, style-transfer

1. Introduction

In 2021, 10 million people were reported to have developed tuberculosis (TB), of whom 1.5 million died. As a result, TB was the second most lethal infectious disease globally, ranking above HIV/AIDS and just below COVID-19 [1]. The majority of TB cases occur in developing nations where access to public health care is limited by complex socio-economic factors, making it difficult to identify and control the spread of the disease and resulting in patients receiving improper care [2].

Whilst published research covering cough classification is currently limited, a few studies have shown promising results when distinguishing between: wet and dry coughs [3, 4], pneumonia [5, 6], and more recently COVID-19 [7, 8, 9]. Because TB is predominately a respiratory disease, it results in patients developing a chronic cough. It has been shown in previous work that it is possible to distinguish between the coughs of TB patients and healthy controls by utilising simple statistical classifiers [10]. More recently, these methods have been evaluated on a dataset that aims to reflect *real-world* conditions, whereby coughers all suffer from some lung ailment that is in some cases TB [11]. Whilst frequency bands important for classification were identified [10], a thorough investigation into the acoustic patterns being learnt has not yet been conducted. Moreover, work considering TB cough classification has relied on linear models utilising fixed dimensional inputs which are typically frame-wise averages of acoustic features. Thus, temporal information present in a cough has so far been disregarded.

In this work we show that recurrent deep learning architectures can be used successfully for TB cough classification, and improve upon existing methods. In addition, by incorporating an attention mechanism and a new loss term, combined with frugal feature selection, we show that model generalisation can be improved. Using the same attention mechanism, we are able to visualise the temporal regions of the feature space that are learnt to be important for cough classification. By considering idealised TB negative and TB positive coughs produced by a technique normally used for neural style transfer, we discuss the distinct characteristics of a TB cough captured by the neural network.

2. Data

We report classification results on a dataset comprising 74 individual patients and 1564 coughs. Previous work in TB cough classification has relied on relatively small datasets gathered in a single recording environment from a small number of patients. This is problematic when training deep-architectures due to their tendency to overfit, for example, to confounding socio-environmental factors which are especially important to disregard in a clinical setting [12]. Relying on recordings from a single environment restricts data diversity and consequently the final model's ability to generalise. In an attempt to address this, we combine the datasets used previously in [10] and in [11], referred to as the Brooklyn and Wallacedene datasets respectively. This is in an effort to yield a more environmentally diverse dataset. Brooklyn was collected in a noise-isolated facility from patients known to have TB and healthy controls, whilst Wallacendene was collected in a noisy environment, from patients who all suffer from either TB or some other lung ailment (confirmed later by sputum analysis). This combined dataset is summarised in Table 1.

Table 1: *Dataset used for experimentation. TB and $\overline{TB}$ indicate TB positive and negative respectively.*

	TB	$\overline{\text{TB}}$	Total
Patients	28	46	74
Total coughs	844	720	1564
Mean cough length (s)	0.60	0.64	0.62
Std dev cough length (s)	0.34	0.29	0.32

2.1. Cross-validation and testing

We divide the combined dataset into a training set (which is further subdivided for cross-validation) and a test set, containing 49 and 25 patients respectively. Importantly, both the Brooklyn and Wallacedene datasets are represented equally in all splits. Furthermore, splits are performed patient wise, ensuring that all coughs originating from the same patient are only present in one set, and we ensure a uniform distribution of TB positive and negative (TB and $\overline{\text{TB}}$) patients across splits. The training set is further divided into 4 folds (each consisting of its own train and development set) for cross-validation using the same previously described procedure.

3. Models

We first train and evaluate several binary classifiers, including: logistic regression (baseline), a basic BiLSTM and a BiLSTM with attention. Next, we use the attention-based architecture to deduce important cough characteristics by generating idealised coughs for each class through a neural style-transfer technique. A sequential forward search (SFS) [13] is performed for both recurrent architectures to identify the most important frequencies for classification and to investigate its impact on model generalisation. This information is considered in conjunction with the temporal regions identified as important for classification by analysis of the attention weights.

3.1. Logistic regression

Previous work in TB cough classification has focused on simple linear models since it was observed that complex neural networks resulted in degraded performance. In both [10] and [11], logistic regression (LR) outperformed all other considered classifiers. As such, we use it as a baseline with which our architectures will be compared. LR is a simple approach that linearly models a probability $f(\mathbf{x})$ given a set of d predictors $\mathbf{x} \in \mathbb{R}^d$ using learnable parameters $\boldsymbol{\theta}$. This highlights an important limitation of LR: each predictor $\mathbf{x}$ is a feature vector computed from a frame of audio. To obtain the probability that a cough is associated with TB, the average of the frame probabilities is computed. In doing so, any temporal information is lost.

3.2. BiLSTM

RNNs have successfully been used in several acoustic classification tasks. Acoustic feature vectors are processed sequentially, each updating the network's internal hidden states which contain complex context-rich information and are available at the next time step. This allows the network to learn temporal relations important to the task at hand. In this work, we make use of a BiLSTM which extends the LSTM architecture [14] by processing the sequence in both forward and backward temporal directions.

A high-level diagram of the network is shown in Figure 1. The single BiLSTM layer has a 32-dimensional hidden state whereby the final outputs in both directions are concatenated to form $\mathbf{q}$. This embedding is then passed through a small feed-forward network with a 32-dimensional hidden layer and ReLU activations followed by an output layer. We include dropout before the first linear layer with a probability of 0.5. Lastly, to account for the unbalanced nature of our dataset, we use weighted cross-entropy as our loss function.

3.3. BiLSTM-Att

The development of the attention mechanism [15] has revolutionised deep learning research. With a focus on acoustic classification, attention-based architectures achieve near state-of-the-art results on tasks such as the Google speech commands dataset [16, 17]. In addition, the intuitive nature of the architecture allows for analysis of what the network is learning, reducing the black-box notion commonly associated with deep learning.

We develop an attention-based model by integrating an attention layer into the above BiLSTM architecture. Instead of passing $\mathbf{q}$ directly to the fully connected network as is done in the basic BiLSTM architecture, the attention mechanism uses $\mathbf{q}$ as the query and outputs a weighted average (by the attention score) of all the BiLSTM outputs, thereby allowing the single output vector to capture information from the temporal regions most relevant for classification and suppress information from unimportant regions in time.

We design this architecture bearing in mind the fact that it will be used to aid in the understanding of the acoustic signature of a TB cough. Accordingly, a new loss term is introduced that encourages the embedding layer of the network (the output of the attention block) to generalise across patients of the same TB status. This is performed to inhibit our subsequent model analysis to be confounded by attributes learned irrelevant to TB cough classification, namely patient identity, an attribute present in cough [18]. This is accomplished by incorporating a GE2E loss term which was originally proposed to determine speaker embeddings by encouraging the network to keep embeddings close together when from the same target speaker, and further apart for different speakers [19]. We consider TB and $\overline{\text{TB}}$ coughs to represent two respective "speakers". Hence the similarity between the embedding centroids of different patients with the same TB status is maximised, whilst minimising the similarity between embeddings of the TB and $\overline{\text{TB}}$ classes. The combined loss function used to train our network is given in Equation 1, where B is the batch size.

$$\mathcal{L} = -\frac{1}{B}\left(\sum_{b}^{B} \beta \cdot \mathbf{y_b} \cdot \log(\hat{\mathbf{y}}_\mathbf{b}) + \alpha \sum_{j,i} L_{GE2E}(\mathbf{e}_{ij})\right) \quad (1)$$

Here the first term is standard weighted cross-entropy where $\mathbf{y_b}$ and $\hat{\mathbf{y}}_\mathbf{b}$ are the vectors of ground truth and predicted probabilities respectively (where the dimension is the number of classes i.e. two) for a given cough in the batch and β is a vector of class weights (constant throughout training). The weight for the under-sampled class is set to 1, while for the over-sampled class it is the ratio of its occurrence in the training set to the total number of samples. In the second term, α is a regularisation parameter, L_{GE2E} is the function that computes the GE2E loss for a specific embedding in a given batch, and $\mathbf{e}_{ij}$ is the embedding vector of the i^{th} cough from the j^{th} class.

4. Experimental procedure

With the exception of LR, which is trained using the standard `scikit-learn` recipe [20], models are trained for 15 epochs with a learning rate of 1×10^{-4} and batch size of 128. After training all 4 folds, the mean development AUC is computed for each epoch. The models from the epoch with the highest mean development AUC are selected, and at test time are ensembled. The decision threshold used for classification was $\bar{\gamma}$, the mean of the decision thresholds γ_n that result in the EER for each fold. Hence the decision threshold was chosen on the basis of the EER as in previous work [10, 11]. We note that it might be possible to improve performance if a strategy that chooses this threshold to optimise, for example, sensitivity and specificity, is adopted. However, we leave this investigation for future work.

4.1. Data Augmentation and feature extraction

We experimented with 3 data augmentation techniques: SpecAugment [21], random insertions and deletions, and speed-perturbation [22]. Initial experiments indicated that only speed perturbation was effective, and hence report only this form of

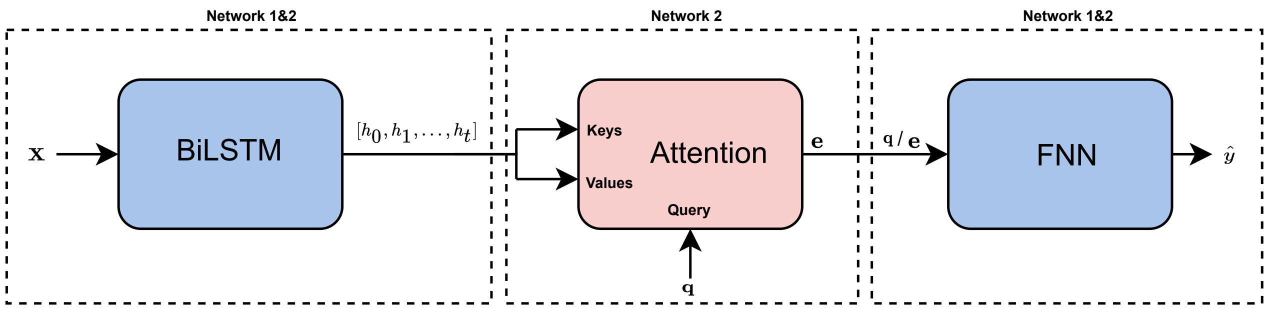
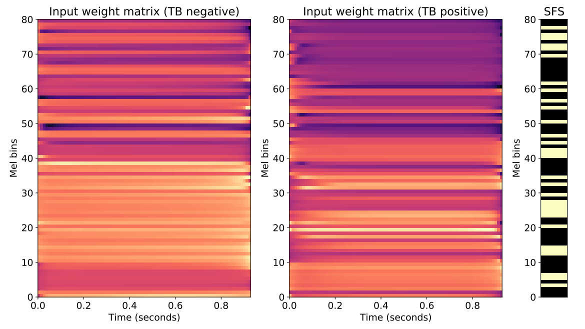


Figure 1: *Structure of the basic BiLSTM (Network 1) and its attention variant (Network 2), with shared components indicated.*

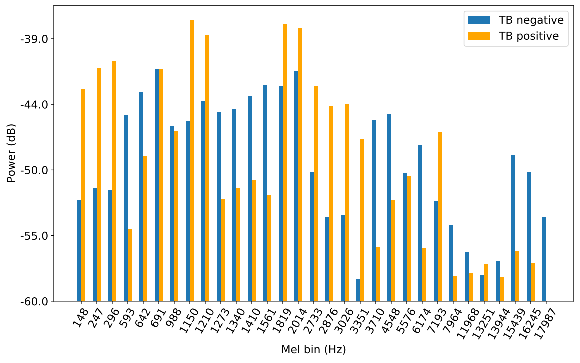
augmentation. We use warping factors of $\{0.9, 1.0, 1.1\}$ resulting in a 3-fold increase in the size of the dataset. We considered three types of acoustic feature: mel-spectrograms, linear filter-bank energies, and MFCCs (with appended velocity and acceleration, as well as cepstral mean and variance normalisation). The former was found to perform best for all architectures, with the ideal number of filter banks being 180, 128, and 80 for LR, the BiLSTM and BiLSTM-Att model respectively. Analysis was based on 2048-sample frames, with successive frames overlapping by 1536 samples (i.e. a frame-skip of 512). Variations in frame length and frame-skip were not considered in this work. All recordings were down-sampled to 44.1kHz before feature extraction.

4.2. Idealised coughs through neural style transfer

In an attempt to understand what the network is learning in order to distinguish between TB and $\overline{\text{TB}}$ coughs, we employ a technique similar to that used in neural style transfer [23] to synthesise an *idealised* cough for each class. This is accomplished by first defining a 2D parameter matrix $\hat{\mathbf{x}} \in \mathbb{R}^{n\times d}$ (initialised to zeros) that will represent the input to the network, where d is the size of the acoustic feature vector seen by the network during training, and n is the number of frames. In this case we select $n = 80$. Next, we perform training as before, but instead of fixing the network input and output and optimizing the weights, we fix the weights and train $\hat{\mathbf{x}}$. This allows the discovery of the input $\hat{\mathbf{x}}$ that best leads to the output class y for the trained weights, i.e. an idealised input cough feature representation for the output class in question.



(a) *Idealised mel-spectrograms for TB negative (left) and positive (center) coughs, with bins identified by SFS shown in yellow (right).*



(b) *Mean power of each idealised mel-spectrogram for the frequency bins deemed most important by SFS.*

Figure 2: *Idealised mel-spectrograms and mean spectral power.*

5. Experimental results and discussion

We present classification performance for the various classifiers investigated and discuss our findings. In addition, we analyse the idealised cough mel-spectrograms produced when applying our adaptation of neural style transfer and observe the attention weights to infer the spectral and temporal regions that the classifier finds most useful for classification.

5.1. Classification

We present development set performance for each considered architecture in Table 2 and the associated test set performance in Table 3. A substantial increase in classification performance over the LR baseline is observed with regards to the basic BiLSTM model with all metrics either matched or improved upon, most notably the specificity. The test set AUC for both the basic BiLSTM and its attention variant are comparable, but there is a large discrepancy in the remaining metrics. This indicates that the decision threshold was not optimal and more robust alternatives to using the EER should be explored in future work.

When inspecting the effect of applying SFS on the two deep architectures, interesting observations can be made. We note a substantial reduction in the standard deviation of the EER-based thresholds determined for the BiLSTM-Att architecture (0.175 before and 0.070 after SFS) whilst the opposite is observed for the BiLSTM without attention. Despite achieving the highest test AUC, an increase in decision threshold standard deviation was observed (from 0.108 to 0.155). An increase in the variability of the decision threshold between folds indicates poorer generalisation. Conversely, with the BiLSTM-Att architecture, better generalisation across the folds is observed which is evident in the reduced standard deviation of the decision threshold. This is especially important with the implementation of a TB screening tool in mind, where model generalisation will be key.

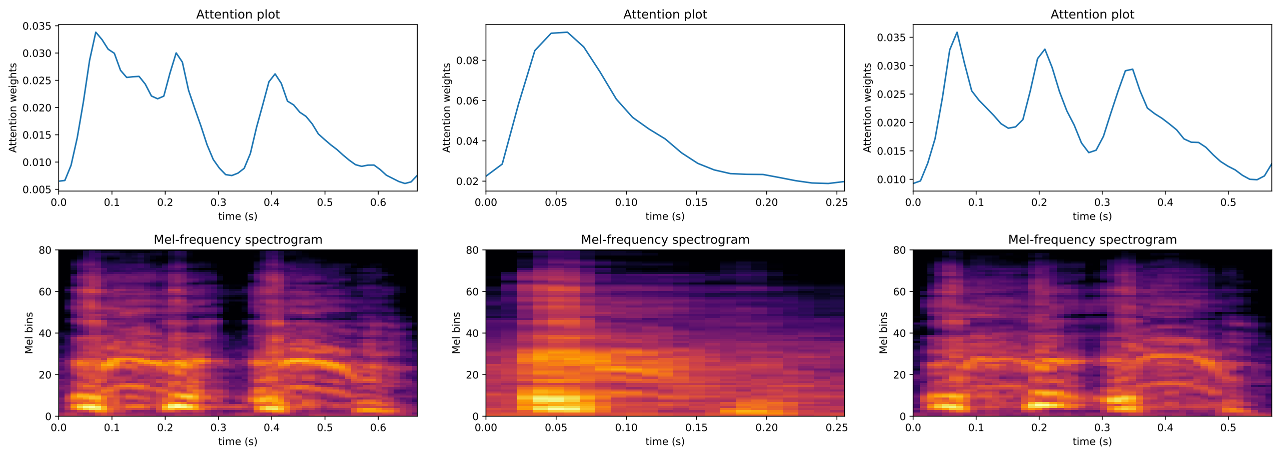


Figure 3: *Various cough mel-spectrograms, all of which originate from the same patient, and their respective attention weights shown above them.*

Table 2: *Mean and standard deviation of the area under the ROC curve (AUC) and the EER decision thresholds ($\bar{\gamma}$) observed during 4-fold cross validation.*

Model	$\bar{\gamma}$	AUC
LR (*baseline*) [10, 11]	0.272 ± 0.103	0.701 ± 0.127
BiLSTM	0.534 ± 0.108	0.777 ± 0.094
BiLSTM (SFS)	0.603 ± 0.155	0.919 ± 0.081
BiLSTM-Att	0.460 ± 0.175	0.873 ± 0.054
BiLSTM-Att (SFS)	0.568 ± 0.070	0.900 ± 0.092

Table 3: *Test set performance for the models listed in Table 2, evaluated through various metrics: sensitivity, specificity, accuracy and area under the curve.*

Model	Sens	Spec	Acc	AUC
LR (*baseline*) [10, 11]	**0.889**	0.625	0.720	0.769
BiLSTM	**0.889**	0.750	**0.800**	0.821
BiLSTM (SFS)	0.667	0.750	0.720	**0.862**
BiLSTM-Att	0.778	0.625	0.680	0.822
BiLSTM-Att (SFS)	0.778	**0.813**	**0.800**	0.850

5.2. Analysis and interpretation

Figure 2a depicts the idealised coughs synthesised by the BiLSTM-Att network using the neural style transfer method described in Section 4.2. Clear differences between idealised $\overline{\text{TB}}$ and TB cough mel-spectrograms are observed. This is further illustrated by comparing the mean power of these idealised coughs as a function of the frequencies determined to be most important by SFS, as shown in Figure 2b. For the idealised TB cough, we observe generally higher power at lower frequencies ($< 500\text{Hz}$) and the mid-band range of $1.8\text{kHz} - 3.3\text{kHz}$ whereas the $\overline{\text{TB}}$ cough has higher power between $1.2\text{kHz} - 1.8\text{kHz}$ and frequencies greater than 3.7kHz, which include frequencies far outside the typical range of human speech ($> 8\text{kHz}$). In Figure 3 we plot the attention weights as a function of time for three cough mel-spectrograms. We observe large importance being placed on regions where the signal has a high power and a large bandwidth, which coincide with the initial bursts of energy for each coughing episode. Whilst only three examples are shown, these observations were made in general. This high energy portion of the coughing sound originates from the lung itself, in particular, the bronchi [24]. It therefore appears that, whilst TB can manifest in all regions of the respiratory tract, the model is relying on some change in the sound produced inside the lungs of TB and $\overline{\text{TB}}$ patients. Further research is necessary to deduce what the physiological causes of this difference in the audio signals could be.

6. Conclusion

In this work, we explored the use of recurrent networks for TB cough classification and use these trained networks to identify and interpret important cough characteristics in both frequency and time. A BiLSTM architecture is shown to improve on previous research, achieving a sensitivity and specificity of 0.89 and 0.75 respectively. This indicates that deeper architectures are viable for TB cough classification, and can improve upon previous state-of-the-art for TB screening. Furthermore, we show that by incorporating frugal feature selection our proposed attention-based architecture exhibits substantially better generalisation across folds than the other considered architectures. This is an important observation for future work, in which datasets will include many more recording domains and associated variability. Utilizing an attention architecture, the importance of certain temporal regions in the cough signal could be visualised. It was observed that the initial voiced regions of cough were the most important for classification. Moreover, by employing a neural style transfer technique, idealised TB negative and positive coughs were synthesised. Subsequent inspection revealed stark differences between energy content in specific frequency bands. In addition to providing new insights into the aspects of a tuberculosis cough that are important for classification, this provides evidence that the TB signal being learnt does indeed originate in the lungs. In future work, we look forward to evaluating our architectures on larger datasets currently being collected [25].

7. Acknowledgements

The authors gratefully acknowledge funding from the EDCTP2 programme supported by the European Union (grant RIA2020I-3305, CAGE-TB).

8. References

[1] World Health Organization, "Global tuberculosis report 2021," 2021.

[2] N. Foster, A. Vassall, S. Cleary, L. Cunnama, G. Churchyard, and E. Sinanovic, "The economic burden of TB diagnosis and treatment in South Africa," *Social science & medicine*, vol. 130, pp. 42–50, 2015.

[3] Y. A. Amrulloh, D. A. Wati, F. Pratiwi, and R. Triasih, "A novel method for wet/dry cough classification in pediatric population," in *2016 IEEE Region 10 Symposium (TENSYMP)*, 2016, pp. 125–129.

[4] V. Swarnkar, U. R. Abeyratne, A. B. Chang, Y. A. Amrulloh, A. Setyati, and R. Triasih, "Automatic identification of wet and dry cough in pediatric patients with respiratory diseases," *Annals of biomedical engineering*, vol. 41, pp. 1016–1028, 2013.

[5] Y. Amrulloh, U. Abeyratne, V. Swarnkar, and R. Triasih, "Cough sound analysis for pneumonia and asthma classification in pediatric population," in *2015 6th International Conference on Intelligent Systems, Modelling and Simulation (ISMS'15)*, 2015.

[6] U. R. Abeyratne, V. Swarnkar, A. Setyati, and R. Triasih, "Cough sound analysis can rapidly diagnose childhood pneumonia," *Annals of biomedical engineering*, vol. 41, pp. 2448–2462, 2013.

[7] 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*, 2021, epub.

[8] P. Mouawad, T. Dubnov, and S. Dubnov, "Robust detection of COVID-19 in cough sounds," *SN Computer Science*, vol. 2, pp. 1–13, 2021.

[9] P. Bagad, A. Dalmia, J. Doshi, A. Nagrani, P. Bhamare, A. Mahale, S. Rane, N. Agarwal, and R. Panicker, "Cough against COVID: Evidence of COVID-19 signature in cough sounds," *arXiv preprint arXiv:2009.08790*, 2020.

[10] G. Botha, G. Theron, R. Warren, M. Klopper, K. Dheda, P. Van Helden, and T. Niesler, "Detection of tuberculosis by automatic cough sound analysis," *Physiological Measurement*, vol. 39, 2018.

[11] M. Pahar, M. Klopper, B. Reeve, G. Theron, R. Warren, and T. Niesler, "Automatic cough classification for tuberculosis screening in a real-world environment," *Physiological Measurement*, vol. 42, 2021.

[12] C. J. Kelly, A. Karthikesalingam, M. Suleyman, G. Corrado, and D. King, "Key challenges for delivering clinical impact with artificial intelligence," *BMC medicine*, vol. 17, pp. 1–9, 2019.

[13] F. J. Ferri, P. Pudil, M. Hatef, and J. Kittler, "Comparative study of techniques for large-scale feature selection," in *Machine Intelligence and Pattern Recognition*. Elsevier, 1994, vol. 16, pp. 403–413.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–1780, 1997.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[16] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," *arXiv preprint arXiv:2005.06720*, 2020.

[17] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," in *INTERSPEECH*, 2020.

[18] M. Zhang, Y. Chen, L. Li, and D. Wang, "Speaker recognition with cough, laugh and 'wei'," in *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2017.

[19] L. Wan, Q. Wang, A. Papir, and I. L. Moreno, "Generalized end-to-end loss for speaker verification," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[20] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," in *INTERSPEECH*, 2019.

[22] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *INTERSPEECH*, 2015.

[23] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[24] J. Korpáš, J. Sadloňová, and M. Vrabec, "Analysis of the cough sound: an overview," *Pulmonary pharmacology*, vol. 9, pp. 261–268, 1996.

[25] "Cough Audio triaGE for TB (CAGE-TB)," https://www.cagetb.org/, accessed: 2022-06-19.

Appendix B

Dataset splits

For ease of any other future experiments, we present the patient ID's for the dataset splits (test and development sets) in a list format.

B.1. Fold 1

B.1.1. Test set

[‘Wu0376’, ‘Wu0489’, ‘Wu0426’, ‘Wu0453’, ‘Wu0480’, ‘Wu0427’, ‘Wu0398’, ‘Wu0436’, ‘Wu0401’, ‘Wu0440’, ‘Wu0438’, ‘Wu0449’, ‘CONX_088’, ‘C_429’, ‘CONX_077’, ‘CONX_061’, ‘CONX_081’, ‘CONX_085’, ‘C_443’, ‘C_430’, ‘C_412’, ‘C_438’, ‘C_442’, ‘CONX_071’, ‘CONX_065’]

B.1.2. Inner fold 1 development set

[‘Wu0431’, ‘Wu0473’, ‘Wu0459’, ‘Wu0423’, ‘Wu0384’, ‘Wu0378’, ‘CONX_064’, ‘CONX_063’, ‘CONX_068’, ‘CONX_069’, ‘C_410’, ‘C_415’, ‘C_449’]

B.1.3. Inner fold 2 development set

[‘Wu0454’, ‘Wu0442’, ‘Wu0437’, ‘Wu0476’, ‘Wu0393’, ‘Wu0448’, ‘CONX_060’, ‘CONX_086’, ‘CONX_089’, ‘CONX_070’, ‘C_440’, ‘C_407’]

B.1.4. Inner fold 3 development set

[‘Wu0485’, ‘Wu0443’, ‘Wu0479’, ‘Wu0475’, ‘Wu0435’, ‘Wu0380’, ‘CONX_066’, ‘CONX_072’, ‘CONX_082’, ‘C_433’, ‘C_432’, ‘C_425’]

B.1.5. Inner fold 4 development set

[‘Wu0482’, ‘Wu0481’, ‘Wu0486’, ‘Wu0424’, ‘Wu0404’, ‘Wu0388’, ‘CONX_076’, ‘CONX_087’, ‘CONX_083’, ‘C_451’, ‘C_431’, ‘C_418’]

B.2. Fold 2

B.2.1. Test set

[‘Wu0393’, ‘Wu0431’, ‘Wu0482’, ‘Wu0479’, ‘Wu0435’, ‘Wu0475’, ‘Wu0380’, ‘Wu0486’, ‘Wu0384’, ‘Wu0459’, ‘Wu0448’, ‘Wu0485’, ‘CONX_060’, ‘CONX_087’, ‘C_433’, ‘C_440’, ‘C_451’, ‘CONX_076’, ‘C_410’, ‘C_415’, ‘CONX_063’, ‘CONX_069’, ‘C_425’, ‘CONX_089’, ‘CONX_070’]

B.2.2. Inner fold 1 development set

[‘Wu0473’, ‘Wu0453’, ‘Wu0442’, ‘Wu0476’, ‘Wu0404’, ‘Wu0438’, ‘CONX_072’, ‘CONX_081’, ‘CONX_068’, ‘CONX_071’, ‘C_438’, ‘C_407’, ‘C_443’]

B.2.3. Inner fold 2 development set

[‘Wu0454’, ‘Wu0401’, ‘Wu0424’, ‘Wu0376’, ‘Wu0440’, ‘Wu0426’, ‘CONX_066’, ‘CONX_086’, ‘CONX_085’, ‘CONX_065’, ‘C_442’, ‘C_430’]

B.2.4. Inner fold 3 development set

[‘Wu0443’, ‘Wu0436’, ‘Wu0398’, ‘Wu0449’, ‘Wu0378’, ‘Wu0427’, ‘CONX_088’, ‘CONX_061’, ‘CONX_082’, ‘C_429’, ‘C_432’, ‘C_449’]

B.2.5. Inner fold 4 development set

[‘Wu0481’, ‘Wu0480’, ‘Wu0423’, ‘Wu0437’, ‘Wu0489’, ‘Wu0388’, ‘CONX_064’, ‘CONX_077’, ‘CONX_083’, ‘C_412’, ‘C_431’, ‘C_418’]

B.3. Fold 3

B.3.1. Test set

[‘Wu0423’, ‘Wu0404’, ‘Wu0424’, ‘Wu0476’, ‘Wu0437’, ‘Wu0388’, ‘Wu0473’, ‘Wu0481’, ‘Wu0443’, ‘Wu0378’, ‘Wu0442’, ‘Wu0454’, ‘CONX_066’, ‘CONX_086’, ‘CONX_064’, ‘C_432’, ‘CONX_072’, ‘CONX_082’, ‘CONX_068’, ‘C_407’, ‘C_431’, ‘C_449’, ‘CONX_083’, ‘C_418’]

B.3.2. Inner fold 1 development set

[‘Wu0431’, ‘Wu0453’, ‘Wu0482’, ‘Wu0435’, ‘Wu0376’, ‘Wu0438’, ‘CONX_077’, ‘CONX_061’, ‘CONX_069’, ‘CONX_071’, ‘C_429’, ‘C_442’, ‘C_415’]

B.3.3. Inner fold 2 development set

[‘Wu0486’, ‘Wu0459’, ‘Wu0401’, ‘Wu0398’, ‘Wu0440’, ‘Wu0427’, ‘CONX_060’, ‘CONX_076’, ‘CONX_081’, ‘CONX_065’, ‘C_433’, ‘C_410’, ‘C_440’]

B.3.4. Inner fold 3 development set

[‘Wu0480’, ‘Wu0489’, ‘Wu0449’, ‘Wu0448’, ‘Wu0426’, ‘Wu0380’, ‘CONX_088’, ‘CONX_089’, ‘CONX_085’, ‘C_412’, ‘C_451’, ‘C_443’]

B.3.5. Inner fold 4 development set

[‘Wu0485’, ‘Wu0479’, ‘Wu0475’, ‘Wu0436’, ‘Wu0393’, ‘Wu0384’, ‘CONX_087’, ‘CONX_063’, ‘CONX_070’, ‘C_438’, ‘C_430’, ‘C_425’]

Appendix C

The limitations of nested cross-validation

Nested k-fold cross-validation has been used in both [2] and [3] as a means to develop and test models. We attempt to show why this severely limits the degrees of freedom when developing models due to an overlap in inner-fold train/dev splits and outer-fold train/test splits. In both works, an outer k-fold loop partitions the data into train and test sets. For each of these partitions, the train set is further partitioned by a nested k-fold loop into k train and development sets. These development sets are used to select model architecture and tune hyper-parameters.

C.1. The problem

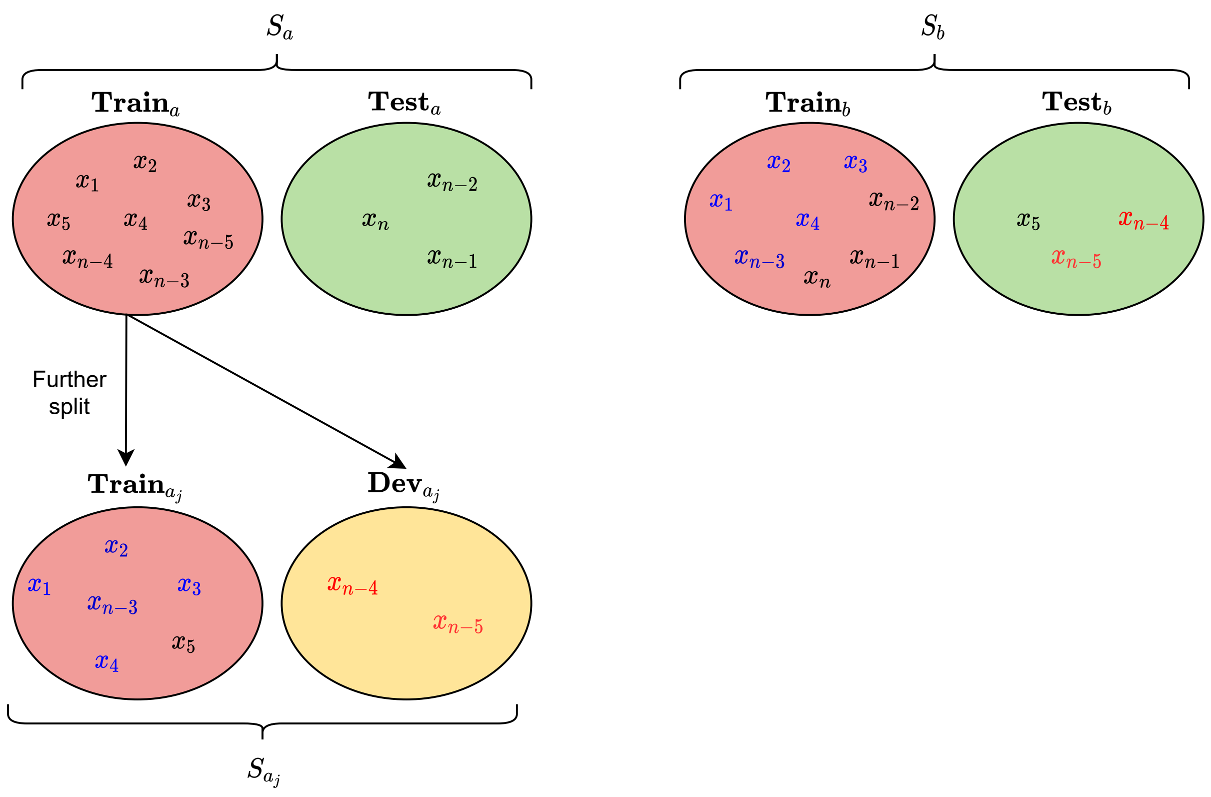
Suppose you have the dataset consisting of n samples:

$$\mathbf{X} = [x_1, x_2, ..., x_n] \tag{C.1}$$

Given two outer-fold splits $\mathbf{S}_a$ and $\mathbf{S}_b$ as shown in Figure C.1, it can be observed that when the training set ($\mathbf{Train}_a$) of $\mathbf{S}_a$ is further partitioned into sub-training ($\mathbf{Train}_{a_j}$) and development sets ($\mathbf{Dev}_{a_j}$), there can be an overlap between $\mathbf{Train}_{a_j}$ and $\mathbf{Train}_b$ and $\mathbf{Dev}_{a_j}$ and $\mathbf{Test}_b$. More concretely:

$$\begin{gathered}\mathbf{Train}_{a_j} \cap \mathbf{Train}_b \neq \emptyset \\ \mathbf{Dev}_{a_j} \cap \mathbf{Test}_b \neq \emptyset\end{gathered} \tag{C.2}$$

In an attempt to quantify this overlap, we derive an expression for the overlap between $\mathbf{Dev}_{a_j}$ and $\mathbf{Test}_b$ given the worst-case scenario that $\mathbf{Dev}_{a_j} \subset \mathbf{Test}_b$. Let the number of samples in the dataset be N, the number of outer folds be k_o and the number of inner folds be k_i. We can state that in a given outer loop split the number of test and train samples respectively are:



S_a
$\mathbf{Train}_a$
$\mathbf{Test}_a$
x_1 x_2 x_3 x_4 x_5 x_{n-5} x_{n-4} x_{n-3}
x_n x_{n-2} x_{n-1}
S_b
$\mathbf{Train}_b$
$\mathbf{Test}_b$
x_1 x_2 x_3 x_4 x_{n-2} x_{n-3} x_n x_{n-1}
x_5 x_{n-4} x_{n-5}
Further split
$\mathbf{Train}_{a_j}$
$\mathbf{Dev}_{a_j}$
x_1 x_2 x_3 x_{n-3} x_5 x_4
x_{n-4} x_{n-5}
S_{a_j}

Figure C.1: Two partitions of a nested k-fold cross validation strategy $\mathbf{S}_a$ and $\mathbf{S}_b$ are visualised. Blue samples indicate training data points that overlap between a nested fold and another outer fold, while red indicates dev/test samples that overlap between the same nested fold and outer fold.

$$\frac{N}{k_o} \tag{C.3}$$

$$N \cdot (1 - \frac{1}{k_o}) \tag{C.4}$$

Thus given a single inner loop split the number of dev samples are:

$$\frac{N}{k_i} \cdot (1 - \frac{1}{k_o}) \tag{C.5}$$

Given the aforementioned worst-case scenario, we can express the overlap between a dev and test set (from two independent outer loops) as:

$$\begin{aligned} &\frac{\frac{\cancel{N}}{k_i} \cdot (1 - \frac{1}{k_o})}{\frac{\cancel{N}}{k_o}} \\ \Rightarrow &\frac{k_o - 1}{k_i} \end{aligned} \tag{C.6}$$

C.2. Discussion

It is clear that what happens in each outer loop is completely independent of each other loop. Optimal hyper-parameters from loop **a** are completely independent of those determined in loop **b**. This is not where we take issue with the relation in Equation C.2. Our main issue is that if one were to ever look at these dev set results (not in the automatic fashion that is used for independent hyper-parameter tuning), and make any general decision based on that (examples to follow), that would be a form of cheating.

But why, each outer loop is completely independent, right? Yes! Cheating does not refer to finding the optimal point of convergence or hyper-parameters for a given model in a given outer loop. But rather, it refers to making global architectural decisions (decisions that apply to all folds) that seem to do better on the dev set (across all outer loops), which in turn due to the relation in Equation C.2 could be construed as cheating - since you are making a decision loosely based off of test-set performance, albeit unintentional. A typical approach taken in model development is to iteratively build an architecture, whereby in each step you quantify its performance on a held-out development set and make adjustments to improve said performance. Such a process can not be followed in nested k-fold cross-validation. Dev set results cannot be used to inform any adjustments or new additions to the model architecture that one is investigating.

If nested k-fold cross-validation is to be used, the model architectures that are to be investigated should be decided upon and **fixed** from the outset of any experiments. It also means that any hyper-parameter tuning being performed by nested k-folds should be completely automatic (and independent between outer loops), since as soon as these dev set results are observed, one would be partly aware of a subset of the test set performance (even though it is from another independent outer loop).

Constraining one's research by fixing your selection of model architectures before any experiments have been run has a high potential to lead to non-optimal solutions. Iterative design is a cornerstone of any good solution, and to cut it out in search of "robust" results seems counterintuitive.

C.2.1. Examples

New architectures: A researcher is investigating two architectures to perform sequence classification: logistic regression and an LSTM. They implement a nested k-fold cross-validation setup since they are working with a very small dataset but still want to present robust results. Before looking at the final test set results (the mean across the outer folds), they have a look at the dev set results (the mean across each outer fold's nested fold means - confusing, I know). Upon looking at the results they notice logistic regression, although much less sophisticated than the LSTM model, seems to perform better. This puzzles them, and they assume it is because they are naively selecting the last output of the LSTM layer to represent the sequence. The researcher recalls an interesting paper they recently read that uses attention to "re-weight" the importance of each output of an LSTM, and combines them into a single context-rich feature vector. The researcher decides it would be a good idea to add this to their vanilla LSTM model and repeat the experiment.

Notice that at first glance, nothing seems out of the ordinary, the researcher simply made an observation and adjusted their architecture accordingly. However, despite never looking at test scores, this decision was made based on a dev set score made up of train/dev splits that partly overlap with train/test splits and has inadvertently cheated.

Feature selection: A research group is utilising a pre-trained multilingual feature extractor (Facebook's wav2vec2-xlsr model) for an acoustic unit discovery model they are currently working on. They utilise a nested k-fold cross-validation strategy to get test scores since they are working with an ultra-low resource language (<1hr of transcribed speech!). It's important for these results to be robust since this system is intended to be deployed in a real-world scenario. One of the hyper-parameters that are automatically tuned during nested k-fold cross-validation is which layer of wav2vec2 to use to extract said features. Due to computational constraints, they can't look at all the layers, so they only consider layers 18-24. After the first outer loop is complete, the eager researchers plot the development ABX scores from that specific outer fold vs the wav2vec2 feature layer number. They notice that the deeper the layer, (that is towards 18), the better the ABX score gets, and at layer 18 it still seems to be decreasing! The researchers stop the experiment, change the config file to consider layers 14-24 and restart the experiment.

Unfortunately caught up in all the excitement, these researchers have made a global decision (applies to all folds) on train/dev splits that partly overlap with another outer loop train/test set and have hence cheated and invalidated all their hard work :(

Appendix D

Wallacedene noise categorisation

Table D.1: Wallacedene cough recording notes per patient, categorised into three disturbance categories: "None", "Some" and "Significant".

ID	Notes	TB status	Disturbance
Wu376	Some microphone disturbance, but it's only between coughs. Minimal background noise.	0	Some
Wu378	Nothing notable. Minimal background noise.	1	None
Wu380	patient sniffs at the end of some coughs. Minimal background noise.	1	None
Wu381	Quite substantial microphone disturbance.	1	Significant
Wu384	Cough is characterised by a hick. Some coughs have background noise (trucks).	0	Some
Wu388	Nothing notable. Some background noise.	1	None
Wu392	Quite substantial microphone disturbance.	1	Significant
Wu393	Nothing notable. Some coughs have significant background noise (trucks, music).	1	Some
Wu398	Nothing notable. Minimal background noise.	0	None
Wu399	Some coughs have microphone disturbances.	0	Significant
Wu401	Nothing notable. Some background noise.	1	None
Wu403	Most coughs have microphone disturbances.	0	Significant
Wu404	Nothing notable. Some background noise.	0	None
Wu405	Coughs towards the end of the recording have microphone disturbances.	0	Significant
Wu413	Quite substantial microphone disturbance.	0	Significant
Wu414	Quite substantial microphone disturbance.	0	Significant
Wu417	A little bit of microphone disturbance, but not as bad as the other coughs	1	Significant
Wu423	Nothing notable. Some background noise.	1	None
Wu424	Nothing notable. A little bit of disturbance between coughs.	0	None

Wu426	Some disturbance, but it is less prominent than other recordings.	1	Some
Wu427	Nothing notable. Some loud speaker background noise.	1	None
Wu430	Some disturbance - gets worse towards the end of the recording.	1	Significant
Wu431	Very slight disturbance.	0	Some
Wu435	Very very slight disturbance.	0	Some
Wu436	Nothing notable. Minimal background noise.	0	None
Wu437	Nothing notable. Minimal background noise.	0	None
Wu438	Very slight disturbance, volume quite low.	0	Some
Wu440	volume quite low.	0	Some
Wu442	Nothing notable. Minimal background noise.	1	None
Wu443	Very forced coughs, rapid breathing. Slight disturbance.	0	Some
Wu448	Coughs are forced. Some background noise.	0	Some
Wu449	Some minimal disturbance. Gets better towards the end.	0	Some
Wu450	Disturbances seem to become more prevalent towards the end.	0	Significant
Wu453	It sounds like something is being brushed against the mic in the beginning.	0	Some
Wu454	Nothing notable. Minimal background noise.	0	None
Wu459	Nothing notable. Minimal background noise.	0	None
Wu471	Disturbance is quite bad in a few coughs.	0	Significant
Wu473	Nothing notable. Minimal background noise.	0	None
Wu475	Slight disturbance.	1	Some
Wu476	Nothing notable. Minimal background noise.	0	None
Wu479	Slight disturbance, but not too notable.	1	Some
Wu480	Slight disturbance.	0	Some
Wu481	Coughs are quite quiet? Shame this man is struggling.	1	Some
Wu482	Nothing notable. Minimal background noise.	0	None
Wu485	Slight disturbance. Gets worse towards the end.	0	Some
Wu486	Nothing notable. Minimal background noise.	0	None
Wu488	Some groovy music.	1	Significant
Wu489	Nothing notable. Minimal background noise.	0	None

Wu494	Disturbance is quite bad in a few coughs.	0	Significant