NANYANG TECHNOLOGICAL UNIVERSITY

SCSE19-0277

ROBUST SPEECH EMOTION RECOGNITION WITH REPRESENTATION LEARNING AND MULTI-TASK LEARNING OF EMOTION AND GENDER CLASSIFIERS

Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Computer Science of the Nanyang Technological University

by

Kennard Chan Yanting

School of Computer Science and Engineering
2020
# Table of Contents

Table of Contents .................................................. 2
Abstract ............................................................... 5
Acknowledgements ...................................................... 5
List of Figures .......................................................... 7
List of Tables ........................................................... 9
Introduction ............................................................ 11
  Motivations ........................................................ 11
  Objectives and Scope ............................................. 11
  Organisations ..................................................... 11
Previous Works ......................................................... 12
  Autoencoder ....................................................... 12
  EmoDB Dataset .................................................... 12
  RAVDESS Dataset ................................................ 13
  PRAAT Features ................................................... 13
  Mel-frequency cepstrum (MFC) and Mel-frequency cepstral coefficients (MFCCs) 13
Speech Emotion Recognition using Representation Learning ........................................... 13
  Experiments on RNN instead of using PRAAT features ................................................. 14
    3.1.1 RNN trained on Input features consisting of Amplitude and Pitch of Audio chunks 14
    3.1.2 RNN trained on Input features consisting of Mel-frequency cepstral coefficients (MFCCs) 15
  Experiments using PRAAT features ................................................................................. 16
    3.2.1 Justification on Using Neural Networks trained on PRAAT features over RNN trained on Input features such as MFCC 16
    3.2.2 Experimental Set-up for obtaining Gender-invariant Representations of input instances 17
    3.2.3 Autoencoders ........................................................................................................ 19
    3.2.4 Supervised Autoencoder and Supervised Loss ...................................................... 20
    3.2.5 Independent Component Analysis (ICA) and Independence Loss ......................... 22
    3.2.6 Triplet Loss .......................................................................................................... 23
      3.2.6.1 Additional Observations on Triplet Loss ......................................................... 26
      3.2.6.2 Combining Triplet Loss, Supervised Loss, and Independence Loss together ........ 28

3.2.7 Deep Embedded Clustering (DEC) 29
3.2.8 Soft Nearest Neighbour Loss (SNNL) 30

Speech Emotion Recognition using MTL 32
  Multi-Task Learning (MTL) 32
    4.1.1 Cross-Stitch Networks 33
    4.1.2 Sluice Networks 34
    4.1.3 Weight Suppression MTL 36
    4.1.4 Input Suppression MTL 37
    4.1.5 Correlated Information Removal MTL 38
    4.1.6 Gender Information Capitalization MTL 40

Further Investigation into Correlated Information Removal MTL in Section 4.1.5 43
  4.2.1 Joint Cost Optimization 44
  4.2.2 Noise Optimization 45
  4.2.3 Inhibited Learning 47
  4.2.4 Exploiting Observations from Correlated Information Removal MTL to design a
  Replacement of Autoencoders (i.e. Replacing Autoencoders with Joint Cost
  Optimization Algorithm) 49
  4.2.5 Investigating Application of Joint Cost Optimization Algorithm on Imbalanced
  Gender Train-Test Sets split 51

Other Techniques that seek to capitalise on the Gender information in Training Data to
improve the Emotion Classification 53
  4.3.1 Female-Male Convergence 53
  4.3.2 Stacked Model 54
    4.3.2.1 Class-based and Re-centered (CAR) Data Augmentation 56
    4.3.2.2 Selective Accelerated Learning (SAL) 56

Combining Datasets of Speeches vocalized in Different Languages 57
  4.4.1 Applying Class-based, Re-centred, Training Set Augmentation technique on
  Datasets of different languages 58
  4.4.2 Applying Selective Accelerated Learning technique on Datasets of different
  languages 59
  4.4.3 Combination of Class-based and Re-centred (CAR) and Selective Accelerated
  Learning (SAL) on Datasets of different languages 60

Performance of Selective Accelerated Learning in Image Domain 62

Evaluation of Techniques against State-of-the-art Performance 65
  Experimental Set-up 65
  Training of the Models 66
  Performance of Gender Classifier stacked on top of Emotion Classifier against the
  State-of-the-art Performance for EMODB 67
  Class-based and Re-centred (CAR) Data Augmentation against the State-of-the-art
  Performance for EMODB 68
  Selective Accelerated Learning against the State-of-the-art Performance for EMODB 69
Combining CAR and SAL against the State-of-the-art 70
Comparison of performances of Stacked Model with CAR and SAL with the existing approaches 71

Conclusions and Future Work 72
Conclusions 72
Recommendation in Future Work 73

Reflection on Learning Outcome Attainment 75
References 76
Abstract

This project is divided into three parts. The first part of this project is to find robust representations of speech signals for predicting emotions. The representations should be invariant to speaker characteristics such as gender of speaker and language of speech. We mainly focus on gender. Different techniques such as Supervised Loss and Triplet Loss will be used to extract robust representations of emotional speeches from audio signals. The second part of this project will investigate the feasibility of combining two different tasks (emotion classification and gender classification) together so as to achieve improvements for both tasks. In other words, we are applying Multi-task learning (MTL) on emotion classification and gender classification for speech signals. The third part and the highlight of this project, is the use of the two techniques - Class-based and Re-centred (CAR) Data augmentation and Selective Accelerated Learning (SAL) - to improve speech emotion recognition. However, due to a pending Technical Disclosure submission, we are unable to present details behind CAR and SAL.
Acknowledgements

First of all, I would like to express my sincere thanks and great gratitude to Professor Jagath for his time and effort to guide me for this Final Year Project. Secondly, I would like to thank Dr Chamara and SCALE lab for the resources they have provided me with to carry out my experiments. Lastly, I would like to thank Nanyang Technological University for this wonderful opportunity to carry out meaningful research.

Kennard Chan
Janaury 2020
List of Figures

* - denote Figures that have to be omitted due to a pending Technical Disclosure submission and a potential Patent application.

Figure 1 - Chart of the Project Schedule
Figure 2a - Illustration of Splitting based on Silence
Figure 2b - Illustration of Splitting based on Fixed Time Interval
Figure 3 - Splitting EmoDB instances into Female Train Set and Male Test Set
Figure 4 - Experimental Set-up for Model to generate Gender-invariant Representations
Figure 5 - Stacked Autoencoder (Model) to generate gender-invariant representation.
Figure 6 - Supervised Autoencoder being used as our Model
Figure 7 - An unsupervised Autoencoder with 1 Hidden Layer, connected to a simple Classifier
Figure 8 - Illustration of the mechanism behind Triplet Loss as presented by Schroff
Figure 9 - Clustering results of the Hidden Activations of the Test instances in each Model
Figure 10 - Clustering results of the Hidden Activations of the Train instances in each Model
Figure 11 - Architecture of Model 1 and Model 2
Figure 12 - Cross-stitch networks for two models as presented by Misra and Shrivastava
Figure 13 - Sluice Network for two models (A and B) as presented by Ruder
Figure 14 - Diagram for Weight Suppression as a means to do MTL.
Figure 15 - Algorithm for Weight Suppression
Figure 16 - Breakdown of the $W_1^{(0)}$ Matrix
Figure 17 - Modified Architecture of Model 1
Figure 18 - The workings of Correlated Information Removal MTL
Figure 19 - Diagram of Gender Information Capitalization MTL
Figure 20 - Workings of Gender Information Capitalization MTL
Figure 21 - Illustration of Model 1 used with Correlated Information Removal MTL
Figure 22 - Algorithm for Joint Cost Optimization
Figure 23 - Algorithm for Noise Optimization
Figure 24 - Intuition behind taking the Differences between weights rather than the Sum.
Figure 25 - Comparison of two different means - (a) and (b) - to identify the important Hidden Neurons in Hidden Layer 2
Figure 26 - Model Architecture used in Inhibited Learning
Figure 27 - Algorithm for Inhibited Learning
Figure 28 - Comparison of the Model Architecture when using AE and using Joint Cost Optimization algorithm.
Figure 29 - Illustration of a Normal Train-Test Sets split and an Imbalanced Gender Train-Test Sets split using EmoDB dataset
Figure 30 - Model Architecture for Female-Male Convergence
Figure 31 - Model Architecture for Stacked Model (Consists of 3 Models)
Figure 32 - Illustration of CAR Part 1 *
Figure 33 - Illustration of CAR Part 2 *
Figure 34 - Model Architecture of Model 2 (Female Classifier) *
Figure 35 - Workings of Selective Accelerated Learning *
Figure 36 - Models used for classifying the EMODB and RAVDESS Datasets
Figure 37 - Splitting of the CIFAR10 Dataset
Figure 38 - Architecture of both CNN 1 and CNN 2
Figure 39 - Architecture of both CNN 1 and CNN 2 when we use the Pre-trained MobileNet.
Figure 40 - Architectures of the Gender Classifier, Male Emotion Classifier and Female Emotion Classifier
Figure 41 - Combined Architecture of the Gender Classifier, Male Emotion Classifier and Female Emotion Classifier i.e. the Stacked Model *
List of Tables

Table 1 - Results of using RNN trained on Audio Chunks in EmoDB
Table 2 - Results of using RNN trained on MFCCs in EmoDB
Table 3 - Results of using RNN trained on group of selected input features in EmoDB
Table 4 - Comparison of RNN models with Fully Connected NN trained on PRAAT features
Table 5 - Results of using only Stacked Autoencoders as our Model for Normal Train-Test Split and Female-Male Train-Test Split of the Dataset
Table 6 - Results of Supervised Autoencoder and Unsupervised Autoencoder
Table 7 - Results of Autoencoder with and without Independence Loss
Table 8 - Results of Combining Supervised Loss and Independence Loss
Table 9 - Results of Model with and without Triplet Loss (Test Set)
Table 10 - Results of Triplet Loss with much larger Triplet Loss Weighing factor, $\alpha$ (Test Set)
Table 11 - Results of Model with and without Triplet Loss (Train Set)
Table 12 - Results of Combining Supervised Loss, Independence Loss, and Triplet Loss
Table 13 - Results for an Autoencoder with and without DEC
Table 14 - Results of combining DEC with Triplet Loss
Table 15 - Results of Model with and without Soft Nearest Neighbor Loss
Table 16 - Results of Cross-stitching the Models together
Table 17 - Results of Sluice Network
Table 18 - Results of Weight Suppression
Table 19 - Results of Input Suppression
Table 20 - Results of Correlated Information Removal MTL
Table 21 - Results of Correlated Information Removal MTL with Supervised Loss and Triplet Loss
Table 22 - Results of Correlated Information Removal MTL with Female-male Train-test split for Training Phase 1
Table 23 - Results of Gender Information Capitalization MTL
Table 24 - Results after 1 iteration of Joint Cost Optimization
Table 25 - Results after 2 iterations of Noise Optimization
Table 26 - Results of using Inhibited Learning
Table 27 - Comparison of AE and Joint Cost Optimization Algorithm
Table 28 - Comparison of AE and Joint Cost Optimization algorithm, using Imbalanced Gender Train-Test Sets split
Table 29 - Results of Female-Male Convergence
Table 30 - Results of Stacked Model
Table 31 - Results of using and not using CAR for Model 3
Table 32 - Results of using Selective Accelerated Learning on Male Model (Model 3)
Table 33 - Results of using Selective Accelerated Learning on Female Model (Model 2)
Table 34 - Results of using a Combination of CAR and SAL
Table 35 - Results of Applying CAR technique on Datasets of different languages
Table 36 - Results of Applying CAR technique on Datasets of different languages on reduced Training Set
Table 37 - Results of Applying SAL technique on Datasets of different languages
Table 38 - Results of using both CAR and SAL techniques on Datasets of different languages
Table 39 - Results of applying SAL on Object Recognition Task
Table 40 - Results of SAL with CNN 2 being initialised with the trained Convolutional Layer of CNN 1
Table 41 - Results of applying SAL on Object Recognition Task with Pre-trained MobileNet features
Table 42a - Results of Stacked Model against the State-of-the-art Performance for EMODBD
Table 42b - Results of Stacked Model against the State-of-the-art Performance for RAVDESS
Table 43a - Effect of CAR on EMODDB
Table 43b - Effect of CAR on RAVDESS
Table 44a - Results of CAR against the State-of-the-art Performance for EMODDB
Table 44b - Results of CAR against the State-of-the-art Performance for RAVDESS
Table 45a - Effect of SAL on EMODDB
Table 45b - Effect of SAL on RAVDESS
Table 46a - Results of SAL against the State-of-the-art Performance for EMODDB
Table 46b - Results of SAL against the State-of-the-art Performance for RAVDESS
Table 47a - Results of CAR and SAL against the State-of-the-art Performance for EMODDB
Table 47b - Results of CAR and SAL against the State-of-the-art Performance for RAVDESS
Table 48 - Comparison of accuracies (%) of speech emotion recognition by Stacked Model with CAR and SAL with existing state-of-art techniques.
Chapter 1

Introduction

This chapter is an introduction of the project. It will explain the motivations, objectives, scope of this project.

1.1 Motivations

In typical machine learning applications, a model is often trained using a pre-collected and prepared dataset. However, when the model is being deployed for client usage, it often possible that the input to the model will be significantly different from the instances in the pre-collected and prepared dataset that had been used to train the model. For example, in the case of a Speech Emotion Recognition model, the audio speeches used to train the model could differ from the audio speeches of the clients in terms of the gender, age, accent of the speech signals. Thus, it is important for our model to find robust representations of the speech signals which are invariant to speaker characteristics.

1.2 Objectives and Scope

The aim of this project is to find robust representations of speech signals for predicting emotions. The representations should be invariant to speaker characteristics such as gender of speaker and language of speech. We mainly focus on gender. In addition, we also hope to investigate techniques that would be useful in improving the accuracy of speech emotion recognition in general. One way is to use gender information to improve emotion recognition in Multi-task learning settings. The other way is to use the two techniques - Class-based and and Re-centred (CAR) Data augmentation and Selective Accelerated Learning (SAL).

1.3 Organisations

The report is organised into 6 Chapters. Chapter 1 introduces the aims of the project. Chapter 2
contains a general literature review of the related fields. Chapter 3 will describe the different techniques, experiments, and results of extracting robust, gender-invariant representations of emotional speeches from audio signals. Chapter 4 will explore the methods and experiments conducted to investigate the feasibility of combining two different tasks (emotion classification and gender classification) together so as to achieve improvements for both tasks. In other words, we are using Multi-task Learning on emotion classification and gender classification on speech signals. Chapter 4 will also describe some of the motivations that led to the discovery of CAR and SAL. Chapter 5 will describe the performance of CAR and SAL against state-of-the-arts techniques. Lastly, Chapter 6 will wrap up the project with the conclusion.

1.4 Project Schedule

The Project Schedule is illustrated below. The schedule was drawn out before the commencement of the research, and there were deviations from the schedule due to the nature of research work. Time beyond 31 Dec 2019 is meant to be spent on the writing of the Interim report and Final report.

Figure 1 - Chart of the Project Schedule
Chapter 2

Background

This chapter will discuss and explain common and popular concepts and techniques that will be used in the later chapters of this report.

2.1 Autoencoder

As Autoencoder [1] is a fairly popular model used in neural networks, we will only be describing it briefly here. An autoencoder is a type of artificial neural network that is trained to attempt to copy its input to its output in an unsupervised manner. Autoencoders aim to learn a representation or encodings for a set of data. Autoencoders are typically used for dimensionality reduction and removal of noise in data.

2.2 EmoDB Dataset

The Berlin Database of Emotional Speech (EmoDB) [2] consists of 500 german speech audio samples spoken by 10 different actors (5 Male and 5 Female). There are 7 Emotion Classes, namely Happy, Angry, Anxious, Fearful, Bored, Neutral or Disgusted. There are 10 different texts that could be spoken by the actors.

2.3 RAVDESS Dataset

The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) Dataset [3] consists of emotional speeches and songs vocalised by 24 professional actors (12 female and 12 male). We use only the emotional speeches in our experiments. The emotion classes in the speeches are neutral, calm, happy, sad, angry, fearful, surprise, and disgust. There are a total of 1440 audio speeches that we could use in our experiments. The language used in the speeches is English.
2.4 PRAAT Features

PRAAT [4] is a free computer software for speech analysis. It was developed by Paul Boersma and David Weenink from University of Amsterdam. PRAAT is able to generate features such as Melspectrogram, Mel-frequency cepstral coefficients (MFCCs), Chroma features, Spectral contrast [5], and Tonnetz. We will refer to these 5 features as PRAAT features hereafter. Note also that each of 5 features have a time axis e.g. MFCC for all the instances in EmoDB is an array of the shape [number_of_instances, number_of_MFCC_coefficients, time_duration_of_signal]. However, we take the mean of MFCC with respect to the axis of ‘time_duration_of_signal’, so the MFCC feature will be a matrix of just [number_of_instances, number_of_MFCC_coefficients]. This is done likewise for the other 4 PRAAT Features. We concatenate the 5 PRAAT features together to form a single multi-dimensional array. Thus, PRAAT Features do not have a time axis.

2.5 Mel-frequency cepstrum (MFC) and Mel-frequency cepstral coefficients (MFCCs)

According to Fourier analysis, any signal can be reduced into a number of discrete frequencies, or a spectrum of frequencies over a continuous range. The power spectrum of a signal describes the distribution of power into the frequency components composing that signal. The Mel-frequency cepstrum (MFC) is a representation of the power spectrum of a soundwave. The MFC is obtained by applying a linear cosine transform of the log power spectrum on a nonlinear mel scale of frequency. The Mel-frequency cepstral coefficients (MFCCs) [6] are coefficients that collectively make up a MFC.
Chapter 3

Speech Emotion Recognition using Representation Learning

This chapter will describe the different techniques, experiments, and results of extracting gender-invariant representations of emotional speeches from audio signals.

3.1 Experiments on RNN instead of using PRAAT features

This section (with its sub-sections) will explore the usefulness of recurrent neural networks (RNN) in conducting emotion recognition in EmoDB dataset. We will explore using RNN models to train on two types of input features: 1. Input features are the Amplitude and Pitch of audio chunks (segments of the signal) 2. Input features are the Mel-frequency cepstral coefficients (MFCCs).

3.1.1 RNN trained on Input features consisting of Amplitude and Pitch of Audio chunks

Each example in the EmoDB dataset is a wav file which records a short speech spoken by an actor. We first split the wav file into audio chunks, and there are two ways we can do that: 1. Splitting based on silence detected in the audio file 2. Splitting based on a fixed time interval. Figure 2a and Figure 2b below illustrates these two different ways to split an audio file into chunks.

![Figure 2a - Illustration of Splitting based on Silence](image-url)
The input features that we will be feeding to a RNN consist of the maximum amplitude and pitch of each audio chunk. For example the $k^{th}$ input example in EmoDB can have 6 audio chunks. This means the length of the time axis for the $k^{th}$ input example will be 6. We compute and collect the maximum amplitude and pitch of the 6 chunks separately. The maximum amplitude is just the maximum amplitude of a chunk. As for the pitch, we take the dominant frequency in that chunk. Additionally, we also included the maximum rate of change in amplitude (i.e. maximum gradient of amplitude) of each chunk. Thus, the total number of input dimensions is 3.

We use a simple RNN with 1 hidden layer. The results for the RNN, with the different chunk-splitting techniques are presented in Table 1 below. We obtain that using audio chunks split based on time interval seems to work better than chunks split by silence. The reason for this could be that chunks which represent complete silence in a segment of an audio signal may prove to be useful. When we split by chunks based on silence, we ignore and drop all the chunks which represent complete silence in the speech. But complete silence or pauses in speech may be useful in conveying emotions in speech. Thus, this could have led to a drop in performance for chunks split based on silence compared to chunks split based on time interval.

Table 1 - Results of using RNN trained on Audio Chunks in EmoDB

<table>
<thead>
<tr>
<th></th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN trained with chunks split based on time interval</td>
<td>0.561</td>
</tr>
<tr>
<td>RNN trained with chunks split based on silence</td>
<td>0.383</td>
</tr>
</tbody>
</table>

1 We include both the maximum amplitude and maximum rate of change in amplitude (i.e. maximum gradient of amplitude) of each chunk.

2 Pitch here refers to the dominant frequency of each audio chunk.
3.1.2 RNN trained on Input features consisting of Mel-frequency cepstral coefficients (MFCCs)

In this section, we do not split each wav file from EmoDB into chunks. Instead, we directly extract 40 MFCC coefficients from the wav file. We fed the MFCC coefficients into a RNN with 1 hidden layer. We find that using just the first 20 time steps from each input example (wav file) gives us the best results in terms of test accuracy. We present our results in Table 2 below.

Table 2 - Results of using RNN trained on MFCCs in EmoDB

<table>
<thead>
<tr>
<th></th>
<th>RNN Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN trained with MFCCs</td>
<td>0.533</td>
</tr>
</tbody>
</table>

From Table 2 above, we observe that MFCC is able to achieve a test accuracy of 53.3%, which is higher than RNN trained with chunks split based on silence but lower than RNN trained with chunks split based on Time Interval as shown previously in Table 1.

The relatively satisfactory performance of the MFCC without chunk-splitting provides some indications that MFCC coefficients could be used to further improve the performance of RNN if we had used audio chunks. Thus, we explore the viability of extracting the MFCC coefficients from audio chunks (instead of from the entire wav file), and then using them as an additional input to a RNN. This means that our RNN is now trained with input features consisting of maximum amplitude, maximum rate of change in amplitude, pitch, and MFCC coefficients of audio chunks. Note that we split the audio chunks based on time interval. We present the results in Table 3 below.

Table 3 - Results of using RNN trained on group of selected input features in EmoDB

<table>
<thead>
<tr>
<th></th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN trained with group of selected input features(^3)</td>
<td>0.682</td>
</tr>
</tbody>
</table>

The result is a further improvement of the RNN test accuracy to 0.682, which is the highest test accuracy achieved by a RNN in this project. From all these experiments in Section 3.1, Section 3.1.1, and Section 3.1.2, we can conclude three things: 1. For a RNN model, it is better to split the Wav files into audio chunks rather than using the entire Wav file directly. 2. It is better to do chunk-splitting based on time interval rather than based on silence. 3. Including MFCC as one of the input features will be beneficial to the RNN model’s performance.

\(^3\) Group of selected input features consists of maximum amplitude, maximum rate of change in amplitude, pitch, and MFCC coefficients of audio chunks.
3.2 Experiments using PRAAT features

In this section (with its sub-sections), we will justify why we will be using neural networks trained on PRAAT features rather than RNN trained on input features such as MFCCs of audio chunks for the rest of the experiments in this project. Then, we will explain how PRAAT features can be used to generate robust, gender-invariant representations of input instances (speech signals). Note that PRAAT features had been explained in a previous section (Section 2.4).

3.2.1 Justification on using NN trained on PRAAT features over RNN trained on input features such as MFCC of chunks

We build a simple fully connected neural network with 1 hidden layer and let it train on PRAAT features. We compare the results obtained with the previous results that we had obtained using the RNNs with 1 hidden layer. The findings are presented in Table 4 below.

Table 4 - Comparison of RNN models with Fully Connected NN trained on PRAAT features

<table>
<thead>
<tr>
<th>Model</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN trained with Chunks splitted based on Time Interval</td>
<td>0.561</td>
</tr>
<tr>
<td>RNN trained with Chunks splitted based on Silence</td>
<td>0.383</td>
</tr>
<tr>
<td>RNN trained on MFCC (no Chunks)</td>
<td>0.533</td>
</tr>
<tr>
<td>RNN trained with group of selected input features$^4$</td>
<td>0.682</td>
</tr>
<tr>
<td>Fully Connected Neural Net trained on PRAAT features</td>
<td>0.809</td>
</tr>
</tbody>
</table>

Referring to Table 4, we see that PRAAT features perform best in terms of test accuracy of the emotion recognition task. Hence, we will be using neural networks trained on PRAAT features rather than RNN models for the rest of the experiments in this project.

3.2.2 Experimental Set-up for obtaining Gender-invariant Representations of audio speech input instances

Now, we will explore using different models and techniques to carry out robust, gender-invariant, emotion recognition using PRAAT features. In other words, we seek to transform the PRAAT features such that it becomes gender-invariant representations of the original input speech signals.

Now, we will describe the experimental set-up. In this section, we continue working primarily with the EmoDB dataset. In order to verify that we have achieved a model that is able to generate

$^4$ As aforementioned, the group of input features refers to the maximum amplitude, maximum gradient of amplitude, pitch, and MFCC coefficients of audio chunks that are split based on time interval.
gender-invariant representations of the input signals, we need to split the EmoDB instances into a Female train set and a Male test set. This is illustrated in Figure 3 below.

Note that the terms ‘Model’ and ‘simple Classifier’ are not interchangeable in this section. We will explain these two terms now. As usual, the Model will only be trained on the train set (contains only instances spoken by female speakers). Ideally, the Model should be able to remove gender information from the train instances such that the hidden representations that the Model generates is gender-invariant or gender-neutral. The gender-invariant hidden representations will then be used as input into a simple Classifier (a neural classifier with 1 hidden layer) which classifies the inputs into the different emotion classes. This set-up is illustrated in Figure 4.

If the Model is successful in generating gender-invariant representations of the input instances, then the Test accuracy (note that the test set contains only male instances) of the simple Classifier should be “reasonably high” (will be defined later) despite the fact that the Model has seen only female and not male instances during training. This is because the Model is trained to remove or ignore gender information from input instances when generating the gender-invariant representations of the input instances. If the Model is unable to produce representations that achieve “reasonably high” test accuracy in the simple Classifier, then this implies that the Model’s representations is biased towards female instances (train instances) and will be ineffective when given male instances as input.

Another way of looking at this is that the Model and the simple Classifier should not perform any differently in terms of test accuracy when given a Female-Male Train-Test Split† compared to a normal Train-Test Split (Female and Male instances present in both Train and Test Sets). This is because a Model capable of generating gender-invariant representations should be blind to whether an instance is male or female. Thus, we can define “reasonably high” test accuracy (mentioned in the previous paragraph) as being the test accuracy achieved by the Model and simple Classifier when trained and tested with a normal Train-Test Split.

![Figure 3 - Splitting EmoDB instances into Female Train Set and Male Test Set](image)

---

† E.g. There are only instances spoken by Female speakers in the training set and only instances spoken by Male speakers in the test set.
3.2.3 Autoencoders

We first attempt to build the Model mentioned in Section 3.2.2 using Autoencoders. The concept of Autoencoder has been briefly explained in Section 2.1. Figure 5 describes a Stacked Autoencoders with 2 hidden layers. In our experiments, we tried varying the number of hidden layers in the Stacked Autoencoder.

For the Stacked Autoencoder in Figure 5, after it has been trained, the hidden activations (of train instances) in Hidden Layer 2 will then be used as inputs to train the simple Classifier, which is just a neural classifier with 1 hidden layer and 1 softmax layer. After the simple Classifier is trained, we will generate the Stacked Autoencoder’s Hidden layer 2 activations of the test instances. The generated hidden activations will be used as inputs to the simple Classifier. The test accuracy score from the simple Classifier is used to determine how well the Model (Stack Autoencoder) has done in terms of generating gender-invariant representations of the input instances.

We repeat the experiment with a Stacked Autoencoders with 1 hidden layer, the procedures are
the same except that we take the hidden representations from Hidden Layer 1 and feed that into the simple Classifier.

Table 5 - Results of using Stacked Autoencoders as our Model for Normal Train-Test Split and Female-Male Train-Test Split of the Dataset

<table>
<thead>
<tr>
<th>Model</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully connected Neural Network with 0 Hidden layer⁶</td>
<td>0.801</td>
</tr>
<tr>
<td></td>
<td>0.510</td>
</tr>
<tr>
<td>Autoencoder with 1 Hidden Layer⁷</td>
<td>0.779</td>
</tr>
<tr>
<td></td>
<td>0.540</td>
</tr>
<tr>
<td>Autoencoder with 2 Hidden Layers⁸</td>
<td>0.493</td>
</tr>
<tr>
<td></td>
<td>0.502</td>
</tr>
</tbody>
</table>

As described in Section 3.2.2, if a Model is able to generate gender-invariant representations, then the test accuracy of the simple Classifier using the Female-Male Train-Test Split will be as high or similar to the test accuracy achieved in a normal Train-Test Split.

In Table 5 above, we report the test accuracies of the simple Classifier with normal Train-Test Split and with Female-Male Train-Test Split. Note that for each model reported in Table 5, we had conducted hyper-parameter tuning using grid search for the number of hidden neurons in each hidden layer.

Although Autoencoder with 2 Hidden Layers has higher test accuracy in Female-Male Train-Test Split compared to its own test accuracy in Normal Train-Test Split, we still favour the Autoencoder with 1 Hidden Layer as it is able to achieve the highest test accuracy in Female-Male Train-Test Split as compared to the other 2 models. Thus, in later experiments, we will be using Autoencoder with 1 Hidden Layer as our baseline model. Also, we will be focusing primarily on Female-Male Train-Test split for our experiments hereafter. (Note that even though Fully connected Neural Network with 0 Hidden Layer has the highest test accuracy of 0.801 in Normal Train-Test Split, it does not perform well in the Female-Male Train-Test Split, which is important since we are trying to obtain Gender-invariant features)

### 3.2.4 Supervised Autoencoder and Supervised Loss

Supervised Autoencoder is a variant of Autoencoder and was described by Lei [7]. We start with

---

⁶ i.e. is equivalent to an Autoencoder with 0 Hidden Layer. Concretely, this means that the input features (PRAAT features) are directly used as input to the simple Classifier.

⁷ With 350 Hidden Neurons

⁸ With Hidden Layer 1: 250 Hidden Neurons; Hidden Layer 2: 400 Hidden Neurons
a typical Autoencoder with 1 hidden layer. Similarly, the hidden representations in the Autoencoder’s hidden layer will be used as input into a simple Classifier (Neural classifier with 1 hidden layer). Now, to build a Supervised Autoencoder from the Autoencoder, we will use the hidden activations in the hidden layer of the Autoencoder to make predictions about the emotion classes (i.e. by attaching the Autoencoder Hidden Layer to a Softmax Layer), as shown in Figure 6 below. These predictions will form what we will term as Supervised Test Accuracy. Note that the simple Classifier, on the other hand, is also making predictions on the Emotion Classes. We will refer to the test accuracy of the simple Classifier as Simple Classifier Test Accuracy.

Figure 6 - Supervised Autoencoder being used as our Model

\[ J_{\text{Joint}} = J_{\text{Rec}} + \alpha \cdot J_{\text{Sup}} \]  \hspace{1cm} (1)

Where \( J_{\text{Joint}} \) is the overall joint loss for training the Supervised Autoencoder; \( J_{\text{Rec}} \) is Reconstruction Loss in the Autoencoder Hidden Layer; \( J_{\text{Sup}} \) is Supervised Loss (i.e. Softmax Loss - obtained in left pink box of Figure 6), and \( \alpha \) is a weighting factor for the Supervised Loss.

With the Softmax Layer (left pink box of Figure 6) attached to the Hidden Layer of the Autoencoder, we are able to get a Softmax Loss. We will refer this Softmax loss as the **Supervised loss** hereafter. Note also that, like typical Autoencoders, the Autoencoder Hidden Layer in Figure 6 is trying to minimize the Reconstruction loss during training.

During training of the Autoencoder’s Hidden Layer, we will seek to minimise a weighted sum of the Supervised loss and the Reconstruction loss. We represent this weighted sum with \( J_{\text{Joint}} \) as defined in Equation 1 above.

Now, we will explain the intuition behind using Supervised Autoencoder and minimising the Supervised loss and Reconstruction loss jointly. The hidden activations/representations learnt by a regular (Unsupervised) Autoencoder are typically the underlying or abstract attributes of the inputs. In order to better direct this representation learning towards representations that are more useful for the emotion recognition task, we add in the Supervised loss to the Reconstruction loss. As explained by Lei [7], if we just ignore Reconstruction loss completely and use only Supervised loss to train the ‘Autoencoder Hidden Layer’ in Figure 6, we will no longer have an Autoencoder but just a fully connected network with 1 Hidden layer and 1 Softmax layer (left
pink box in Figure 6). Training a representation in this fully connected Hidden layer is likely an under-constrained problem, and will find hidden representations that lead to high classification accuracy but fail to find underlying patterns in the data and is thus unlikely to generalize well. On the other hand, training just an Autoencoder (only Reconstruction loss) will find the underlying patterns in data but its hidden representations will not be very useful for predictions. Supervised Autoencoder combines the two losses (Reconstruction Loss and Supervised loss) so as to learn a representation that not only contains the underlying structure of the inputs, but also allows for accurate prediction performance.

We present our results in Table 6 below. Note that we use Female-Male Train-Test split, and we did hyperparameter tuning for the supervised weighting factor (\(\alpha\)) before settling with the optimal value of 0.1 for \(\alpha\). From the results, we observe that the Supervised Autoencoder is able to generate a more gender-invariant representation as compared to a regular Autoencoder (Since the Simple Classifier Test Accuracy improved significantly).

Table 6 - Results of Supervised Autoencoder and Unsupervised Autoencoder

<table>
<thead>
<tr>
<th>Model</th>
<th>Supervised Weighting factor, (\alpha)</th>
<th>Supervised Test Accuracy (See Figure 6)</th>
<th>Simple Classifier Test Accuracy (See Figure 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupervised Autoencoder(^9)</td>
<td>0.0 (i.e. Unsupervised)</td>
<td>-</td>
<td>0.540</td>
</tr>
<tr>
<td>Supervised Autoencoder with 1 Hidden Layer</td>
<td>0.1</td>
<td>0.392</td>
<td>0.559</td>
</tr>
</tbody>
</table>

### 3.2.5 Independent Component Analysis (ICA) and Independence Loss

The aim of Independent component analysis (ICA) is to decompose a multivariate signal into a number of independent non-Gaussian signals. A variant of ICA is the Reconstruction ICA (RICA) introduced by Le [8]. RICA can be implemented with a Neural Network by minimising the Cost Function in Equation 2a below. What RICA is trying to do in Equation 2a is to decompose the input signal \(x\) (PRAAT features in our case) into a total of \(k\) independent components. Equation 2a is first presented by Le in [8]. In our project, we will refer to this Cost Function in Equation 2a as **Independence Loss** \(J_{\text{ICA}}\) hereafter.

**Reconstruction ICA (RICA)**: minimize \(\frac{1}{2m} \sum_{i=1}^{m} ||W^TW_i x^{(i)} - x^{(i)}||^2_2 + \sum_{i=1}^{m} \sum_{j=1}^{k} g(W_j x^{(i)})\) \hspace{1cm} (2a)

Note that in Equation 2a, \(\lambda\) is an arbitrary constant, \(m\) is the number of instances in dataset, \(x^{(i)}\) is the \(i^{th}\) instance in the dataset, \(k\) is the number of independent components, and \(W_j\) is \(j^{th}\) row of the \(W\) weight matrix where \(W \in \mathbb{R}^{k \times n}\). Note also that \(n\) is the number of dimensions in \(x^{(i)}\). And \(g(.) := \log(\cosh(.)).\)

\(^9\) It is simply a regular Autoencoder with 1 Hidden Layer
\[ J_{Joint} = J_{Rec} + \alpha \cdot J_{ICA} \]  

(2b)

Where \( J_{Joint} \) is the overall joint loss for training the Autoencoder with Independence Loss; \( J_{Rec} \) is Reconstruction Loss of the Autoencoder layer; \( J_{ICA} \) is Independence Loss, and \( \alpha \) is a weighting factor for the Independence Loss.

In our experiments, we set up our model similar to what has been depicted in Figure 6, but we do not attach the Autoencoder Hidden Layer to a Softmax Layer. Also, during training of the model, we minimize a weighted sum of Reconstruction Loss and Independence Loss jointly.

Table 7 - Results of Autoencoder with and without Independence Loss

<table>
<thead>
<tr>
<th>Model</th>
<th>ICA Weighting factor, ( \alpha )</th>
<th>Simple Classifier Test Accuracy (See Figure 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder with 1 Hidden Layer</td>
<td>0.0</td>
<td>0.540</td>
</tr>
<tr>
<td>Autoencoder with 1 Hidden Layer + Independence Loss</td>
<td>0.01</td>
<td>0.544</td>
</tr>
</tbody>
</table>

From Table 7, we see that with Independence loss, we are able to improve the simple Classifier Test Accuracy from 0.540 to 0.544. This suggests that Independence Loss may be able to lead to a more Gender-invariant representation of the input signals, but only slightly. Note that the ICA Weighting factor (\( \alpha \)) of 0.01 is selected after doing hyper-parameter tuning for it.

Additionally, we also conducted experiments combining Supervised Loss with Independence Loss. We present the results in Table 8 below. Combining both losses failed to improve the simple Classifier Test Accuracy beyond what Supervised Loss can achieve alone.

Table 8 - Results of Combining Supervised Loss and Independence Loss

<table>
<thead>
<tr>
<th>Model</th>
<th>Simple Classifier Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupervised Autoencoder with 1 Hidden Layer</td>
<td>0.540</td>
</tr>
<tr>
<td>Supervised Autoencoder with 1 Hidden Layer</td>
<td>0.559</td>
</tr>
<tr>
<td>Unsupervised Autoencoder with 1 Hidden Layer, and with Independence Loss</td>
<td>0.544</td>
</tr>
<tr>
<td>Supervised Autoencoder with Independence Loss</td>
<td>0.555</td>
</tr>
</tbody>
</table>

3.2.6 Triplet Loss

Triplet Loss was first introduced by Schroff [9] in 2015. We will explain the concept of Triplet
Loss with reference to our model (a normal Autoencoder), which is also depicted in Figure 7. We want the hidden representations in Model Hidden Layer to be as Gender-invariant as possible. One way to ensure that is to encourage the hidden representations learnt in Model Hidden Layer 1 to be as similar to one another if they belong to the same emotion class and to be as dissimilar to one another if they belong to different emotion classes. This is also the aim of Triplet Loss.

Figure 7 - An unsupervised Autoencoder (Model) with 1 Hidden Layer, connected to a simple Classifier

Figure 8 - Illustration of the mechanism behind Triplet Loss as presented by Schroff in [9]

More concretely, the mechanism behind Triplet Loss is depicted in Figure 8. A Triplet consists of an Anchor, Negative, and a Positive. Anchor refers to the hidden activation (in Model Hidden Layer in Figure 7) of a single training instance (e.g. is in emotion class k) that we are currently looking at. Negative refers to the hidden activation of a training instance that is not in emotion class k. Positive refers to hidden activation of a training instance that is in emotion class k. Triplet Loss aims to both maximize the distance between the Anchor and a Negative, as well as to minimize the distance between the Anchor and a Positive. Note that a Negative that is very near to the Anchor is called a hard Negative, and a Positive that is very far from the Anchor is called a hard Positive. Triplets with either hard Negative or hard Positive are called hard Triplets.

To train with Triplet Loss, mini-batch training will need to be carried out. In each mini-batch of train instances, we will need to identify the triplets that we are going to use. We experimented with two different ways of doing that: 1. Get and use all the possible triplets in the mini-batch 2. Identify and use only the hard triplets.

The Triplet Loss is formally defined by Schroff [9] in Equation 4 below. N is the total number of identified Triples in a mini-batch of train instances. Function $f$ maps an input instance to a hidden activation in Model Hidden Layer of Figure 7. $x_i^a$, $x_i^p$, and $x_i^n$ represents the Anchor, Positive, and Negative for the $i^{th}$ triplet respectively. $\beta$ is an arbitrary constant. In order to train
an Autoencoder using Triplet Loss, we will use the Loss Function defined in Equation 5 below as our Model’s Loss Function \( J_{\text{Joint}} \). We train our Model by minimizing (via gradient descent in neural network) \( J_{\text{Joint}} \) using either 1. all possible triplets and 2. hard triplets. We present our results in Table 9 below. Note that the values 0.2 and 0.25 for the Triplet Loss Weighing factors \( \alpha \) in Table 9 have been picked after hyperparameter tuned such that Simple Classifier Test Accuracy is being maximised. Each model in Table 9 has only 1 hidden layer.

\[
J_{\text{Triplet}} = \frac{N}{i} \left[ \| f(x^a_i) - f(x^p_i) \|^2 - \| f(x^n_i) - f(x^p_i) \|^2 + \beta \right], \tag{4}
\]

\[
J_{\text{Joint}} = J_{\text{Rec}} + \alpha \cdot J_{\text{Triplet}} \quad \tag{5}
\]

Where \( J_{\text{Joint}} \) is the overall joint loss for training the Autoencoder with Triplet Loss; \( J_{\text{Rec}} \) is Reconstruction Loss of the Autoencoder, \( J_{\text{Triplet}} \) is Triplet Loss, and \( \alpha \) is a weighting factor for the Triplet Loss.

**Table 9 - Results of Model with and without Triplet Loss (Test Set)**

<table>
<thead>
<tr>
<th>Model</th>
<th>Triplet Loss Weighing factor, ( \alpha )</th>
<th>Simple Classifier Test Accuracy</th>
<th>Clustering Accuracy (Test Set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder</td>
<td>0.0</td>
<td>0.540</td>
<td>0.388</td>
</tr>
<tr>
<td>Autoencoder with Triplet Loss (Use all possible Triplets)</td>
<td>0.2</td>
<td>0.544</td>
<td>0.456</td>
</tr>
<tr>
<td>Autoencoder with Triplet Loss (Use only hard Triplets)</td>
<td>0.25</td>
<td>0.551</td>
<td>0.418</td>
</tr>
</tbody>
</table>

Looking at Table 9, we observe that, if we use hard triplets, training the Model with Triplet Loss will improve the simple Classifier Test Accuracy from 0.540 to 0.551. However, if we use all possible triplets instead, then the simple Classifier Test Accuracy will increase more modestly to 0.544. Thus, using hard triplets may lead to a more Gender-invariant representations of the input instances as compared to using all possible triplets.

As aforementioned, the purpose of the Triplet loss is to encourage the hidden representations learnt in Hidden Layer 1 to be similar to one another if they belong to the same emotion class and to be dissimilar to one another if they belong to different emotion classes. We can see how well the Triplet loss is doing just that by looking at the clustering accuracy\(^{10} \) of hidden representations (of the instances in the test set) learnt in Model Hidden Layer. We present the clustering accuracies in the last column of Table 9. As expected, the clustering accuracy improves when we use Triplet Loss with either all possible triplets in a mini-batch or only the hard Triplets in the mini-batch. However, we would like to explore if we could increase the clustering accuracy even

\(^{10}\) Definition of clustering accuracy: The hidden representations from all the test instances are clustered into the 7 clusters. We use 7 clusters because there are 7 emotion classes in EmoDB. Now, we effectively have the test instances in one of the 7 clusters. Then, an algorithm is used to repeatedly attempt to match each of the 7 clusters to one of the 7 emotion classes. The matching that led to the most number of test instances being assigned(mapped) to the correct emotion classes is deemed as the correct matching, and the clustering accuracy is simply the number of correctly assigned test instances divided by the total number of test instances.
further and if that would lead to the simple Classifier accuracy improving even further than what has been shown in Table 9. The way to go about doing this is to increase the Weighing factor, $\alpha$, to a larger value. Thus, we experiment with a larger Weighing factor of 1.0 for both the hard triplets and all possible triplets scenarios and present the results in Table 10 below. Each model in Table 10 only has 1 hidden layer.

In Table 10, we observed that increasing the Triplet Loss Weighing factor $\alpha$ to 1.0 have led to a higher clustering accuracy as compared to what we have seen in Table 9. However, the Simple Classifier Accuracy has failed to improve beyond 0.540, which is the test accuracy obtained without using any Triplet Loss at all.

<table>
<thead>
<tr>
<th>Model</th>
<th>Triplet Loss Weighing factor, $\alpha$</th>
<th>Simple Classifier Test Accuracy</th>
<th>Clustering Accuracy (Test Set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder (i.e. no Triplet Loss)</td>
<td>0.0</td>
<td>0.540</td>
<td>0.388</td>
</tr>
<tr>
<td>Autoencoder with Triplet Loss (Use all possible Triplets in Mini-batch)</td>
<td>1.0</td>
<td>0.540</td>
<td>0.506</td>
</tr>
<tr>
<td>Autoencoder with Triplet Loss (Use only hard Triplets in Mini-batch)</td>
<td>1.0</td>
<td>0.529</td>
<td>0.449</td>
</tr>
</tbody>
</table>

Using t-distributed Stochastic Neighbor Embedding (TSNE) function provided by sklearn, we are able to visualize the clustering as well. The Clustering results are shown in Figure 9 below. However, we find it difficult to visually compare the different Clustering results based on the outputs produced by the TSNE function. Comparing the different Clustering results based on clustering accuracy as seen in Table 9 and Table 10 remains a more objective and accurate approach.
Figure 9 - Clustering results of the Hidden Activations of the Test instances in each Model. For (a), we directly cluster the PRAAT input features of the test instances into 7 clusters (since there are 7 emotion classes in EmoDB). For (b), we cluster the hidden activations (of the test instances) from the Model Hidden Layer into 7 clusters. For (c) and (d), we do the same, but with triplet loss in the Model.

3.2.6.1 Additional Observations on Triplet Loss

Previously, in Figure 9 above, we observe that it is not clear whether Triplet Loss has indeed done its job of inducing the hidden representations to cluster into the different emotion classes. This is because the models are trained using the train set and are not aware of the test set, and in Table 9, Table 10, and Figure 9, the results are all generated based on test set. Thus, we decided to show the clustering accuracy and the Clustering results (the TSNE diagrams) for instances in train set.
Table 11 - Results of Model with and without Triplet Loss (Train Set)

<table>
<thead>
<tr>
<th>Model</th>
<th>Triplet Loss Weighing factor, $\alpha$</th>
<th>Clustering Accuracy (Train Set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder (i.e. no Triplet Loss)</td>
<td>0.0</td>
<td>0.407</td>
</tr>
<tr>
<td>Autoencoder with Triplet Loss (Use all possible Triplets in Mini-batch)</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Autoencoder with Triplet Loss (Use only hard Triplets in Mini-batch)</td>
<td>1.0</td>
<td>0.993</td>
</tr>
</tbody>
</table>

Figure 10 - Clustering results of the Hidden Activations of the Train instances in each Model

In Table 11 above, each model has only 1 hidden layer. We observe in this table that with triplet loss, regardless of whether using all possible triplets or hard triplets only, we are able to obtain very high Clustering accuracy of 1.0 and 0.993 respectively. These values are extremely high...
compared to the no triplet loss case of 0.407. This conclusively proves that Triplet loss is indeed capable of clustering the hidden representations of the input instances into emotion classes effectively (but only for training set).

In Figure 10 above, we show the Clustering results (TSNE diagrams) for the hidden representations of Training instances. It is easy to note visually that the hidden representations of the training instances have been clearly clustered into the different emotion classes.

However, there is an issue that has been raised by results in Figure 10. This is because when we compare Figure 9 and Figure 10 together, we notice that there is a huge disparity in the effectiveness of the clustering in the two figures. Recall that our Train Set is female speaker instances and our Test Set is male speaker instances. The reason why we decided to use Triplet Loss is to have the hidden representations in the Hidden Layer of our Model to be as Gender-invariant as possible. Triplet Loss attempts to do that by encouraging the hidden representations learnt in Hidden Layer 1 to be as similar to one another if they belong to the same emotion class and to be as dissimilar to one another if they belong to different emotion classes. But what Figure 9 and Figure 10 have shown is that we can make hidden representations for the Train instances to be clearly clustered into the different emotion classes and yet our Model continues to perform only adequately on the Test Set. This raises a very important question: Is it even possible for our model to learn to classify the Male speaker instances effectively if the model is only given information of the Female speaker instances. The observations in Figure 9 and Figure 10 seem to provide ‘No’ as the answer. In other words, we cannot expect our models to do well on Male Speakers if we feed in Female Speakers and do not provide the concept of “Male” to our model.

3.2.6.2 Combining Triplet Loss, Supervised Loss, and Independence Loss together

Additionally, we also conducted experiments combining Supervised Loss, Independence Loss, and Triplet Loss (Hard Triplets) together. We present the results in Table 12 below. Combining the three losses will achieve a simple Classifier Test Accuracy of 0.544, which is still less than using Supervised Loss alone, which achieved a simple Classifier Test Accuracy of 0.559. We tried with different combinations of the three losses (i.e. Choosing 2 out of 3) and with various different weights for the Supervised Loss, Independence Loss, and Triplet Loss, but almost none of them achieved a simple Classifier test accuracy that is equal or greater than 0.559. The only exception is the combination of using Independence Loss and Triplet Loss (no Supervised Loss). This combination allows us to achieve the score of 0.563. This suggests that in order to obtain a more Gender-invariant representations of the input signals, we could either use supervised loss only, or use a combination of Independence Loss and Triplet Loss. 
<table>
<thead>
<tr>
<th>Model</th>
<th>Simple Classifier Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupervised Autoencoder</td>
<td>0.540</td>
</tr>
<tr>
<td>Supervised Autoencoder (i.e. Autoencoder with Supervised Loss)</td>
<td>0.559</td>
</tr>
<tr>
<td>Unsupervised Autoencoder with Independence Loss</td>
<td>0.544</td>
</tr>
<tr>
<td>Unsupervised Autoencoder with Triple Loss using Hard Triplets only</td>
<td>0.551</td>
</tr>
<tr>
<td>Supervised Autoencoder with Independence Loss</td>
<td>0.555</td>
</tr>
<tr>
<td>Supervised Autoencoder with Triplet Loss</td>
<td>0.544</td>
</tr>
<tr>
<td>Unsupervised Autoencoder with Independence Loss and Triplet Loss</td>
<td>0.563</td>
</tr>
<tr>
<td>Supervised Autoencoder with Independence Loss and Triplet Loss</td>
<td>0.544</td>
</tr>
</tbody>
</table>

#### 3.2.7 Deep Embedded Clustering (DEC)

Deep Embedded Clustering (DEC) was introduced by Xie [10] in 2016. In a problem of clustering a set of \( n \) data points into \( k \) clusters, instead of clustering directly in the data space, we can first transform the \( n \) data points into a feature space. The mapping of the \( n \) data points from data space to feature space can be done by an Autoencoder. DEC is an algorithm which aims to simultaneously learn a set of \( k \) cluster centers in the feature space and the weights of the Autoencoder that maps data points from data space to feature space. DEC is similar to Triplet Loss in the sense that both techniques are trying to cluster the encodings (i.e. hidden activations) into clearly defined and distinguished clusters.

We conducted experiments with DEC. First, we trained the Autoencoder (with 1 hidden layer), then we run the DEC algorithm (i.e. learn the cluster centers in feature space and update the weights of the Autoencoder simultaneously). We present the results in Table 13 below. Note that each model in Table 13 has only 1 hidden layer.
### Table 13 - Results for an Autoencoder with and without DEC

<table>
<thead>
<tr>
<th>Model</th>
<th>Simple Classifier Test Accuracy</th>
<th>Clustering Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder</td>
<td>0.540</td>
<td>0.388</td>
</tr>
<tr>
<td>Autoencoder with DEC</td>
<td>0.551</td>
<td>0.392</td>
</tr>
</tbody>
</table>

In Table 13, we observe that the simple Classifier Test Accuracy improved from 0.540 to 0.551 when we use DEC. This implies that DEC may lead to a more gender-invariant representations of the input signals.

As both DEC and Triplet loss works using similar ideas, we should not expect a combination of DEC and Triplet loss to lead to a further improvement of the simple Classifier Test Accuracy. This is supported by the results shown in Table 14 below. Note that each model in Table 14 only has 1 hidden layer. The results in Table 14 show that combining both DEC and Triplet loss will not outperform a model which uses DEC and Triplet loss individually.

### Table 14 - Results of combining DEC with Triplet Loss

<table>
<thead>
<tr>
<th>Model</th>
<th>Simple Classifier Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder (i.e. no DEC)</td>
<td>0.540</td>
</tr>
<tr>
<td>Autoencoder with DEC</td>
<td>0.551</td>
</tr>
<tr>
<td>Autoencoder with Triple Loss(^{11})</td>
<td>0.551</td>
</tr>
<tr>
<td>Autoencoder with DEC and Triple Loss(^{12})</td>
<td>0.551</td>
</tr>
</tbody>
</table>

3.2.8 **Soft Nearest Neighbour Loss (SNNL)**

Soft Nearest Neighbour Loss (SNNL) is first introduced by Salakhutdinov [11]. Here, we provide an intuitive illustration of SNNL. Each instance’s hidden activation can be thought as a single point in some high dimensional space. Given the hidden activations of a mini-batch of instances, we imagine if we were to sample a neighboring point \(j\) for every point \(i\) in the mini-batch, where the probability of sampling neighboring point \(j\) depends on the distance between points \(i\) and \(j\). The soft nearest neighbor loss is the negative log probability of sampling a neighboring point \(j\) from the same class as \(i\). Equation 6 shows the SNNL function, as defined by Froost in [12]. In Froost’s definition of SNNL, he also introduced a new hyper-parameter, the temperature, \(T\), to control the relative importance given to the distances between pairs of points.

As suggested by Froost [12], SNNL has the potential to improve learned representations as it

\(^{11}\) Triplet Loss using Hard triplets only

\(^{12}\) Similarly, Triplet Loss using Hard triplets only
works to disentangle the hidden representations of different classes in the hidden layers. In this sense, SNNL is similar to Triplet Loss.

**Definition.** The soft nearest neighbor loss for a mini-batch of \( b \) samples \((x,y)\) at temperature \( T \) is:

\[
J_{SNNL}(x,y,T) = -\frac{1}{b} \sum_{i=1}^{b} \log \left( \frac{e^{-\frac{\|x_i - y_i\|^2}{T}}}{\sum_{k 
eq i} e^{-\frac{\|x_i - y_k\|^2}{T}}} \right)
\]  

(6)

\[
J_{\text{Joint}} = J_{\text{Rec}} + \alpha \cdot J_{SNNL}
\]  

(7)

Where \( J_{\text{Joint}} \) is the overall joint loss for training the Autoencoder with Soft Nearest Neighbour Loss; \( J_{\text{Rec}} \) is Reconstruction Loss of Autoencoder layer; \( J_{SNNL} \) is Soft Nearest Neighbour Loss defined in Equation (6), and \( \alpha \) is a weighting factor for the Soft Nearest Neighbour Loss.

Note that in Equation 6, the \( x \) is a matrix of hidden representations, \( y \) is a vector of class labels. In our experiments, we add SNNL to the training of the hidden layer to see if SNNL is able to outperform Triplet Loss in terms of generating robust, gender-invariant representations of the input instances. Concretely, we are using the cost function in Equation 7 to train our hidden layer.

We present our results on SNNL in Table 15 below. Note that each model in this Table only has 1 hidden layer.

**Table 15 - Results of Model with and without Soft Nearest Neighbor Loss**

<table>
<thead>
<tr>
<th>Model</th>
<th>SNNL Weighing factor, ( \alpha )</th>
<th>Simple Classifier Test Accuracy</th>
<th>Clustering Accuracy (Test Set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder</td>
<td>0.0</td>
<td>0.540</td>
<td>0.388</td>
</tr>
<tr>
<td>Autoencoder with SNNL</td>
<td>0.75</td>
<td>0.555</td>
<td>0.399</td>
</tr>
<tr>
<td>Autoencoder with Triplet Loss(^{13})</td>
<td>-</td>
<td>0.551</td>
<td>0.418</td>
</tr>
</tbody>
</table>

From Table 15 above, we observe that with SNNL, the Simple Classifier Test Accuracy improved to 0.555, which is slightly better than the improvement observed using Triplet Loss (0.551). This suggests that SNNL could be used as a comparable if not better alternative to Triplet Loss.

As expected, the clustering accuracy also increased when we used SNNL (since SNNL works to disentangle the representations of different classes in hidden layers). Also, note that the values 0.75 for the Weighing factor (\( \alpha \)) in Table 15 have been picked after hyperparameter tuning such that Simple Classifier Test Accuracy is being maximised.

\(^{13}\)Triplet Loss using Hard triplets only
It could be interesting to explore whether a combination of Triplet Loss and SNNL would further improve the results. But theoretically, there should not be much improvement in results. This is due to Triplet Loss and SNNL both serving the same function of trying to disentangle the hidden representations of different classes. We conducted experiments which verified that a combination of Triplet Loss and SNNL will not further improve the results.
Chapter 4

Speech Emotion Recognition using MTL

This Chapter will explore the methods and experiments conducted to investigate the feasibility of combining two different tasks (specifically emotion classification and gender classification) together so as to achieve improvements for both tasks. This is also known as Multi-task learning (MTL).

Unlike the previous chapter, we no longer put all our female instances into the train set and all our male instances into the test set. We simply use a typical train and test sets whereby both female and male instances can appear in the train and test sets.

4.1 Multi-Task Learning (MTL)

In this section, we explore the feasibility and usefulness of training two separate models - an Emotion Classifier and a Gender Classifier - simultaneously such that the two models attempt to complement and improve each other during the training. The architecture of the two models (Model 1 and Model 2) that are used in this section and its subsections are shown in Figure 11 below. Note that Hidden Layer 1 and Hidden Layer 2 for both Model 1 and Model 2 are autoencoder layers.

Figure 11 - Architecture of Model 1 and Model 2
In order to combine these two models together and increase both of their performances, we employ the concepts and techniques of MTL. Specifically, we make use of the MTL techniques. The first MTL technique that we will explore is the Cross-Stitch Networks.

### 4.1.1 Cross-Stitch Networks

Cross-Stitch Networks are introduced by Misra and Shrivastava [13]. Cross-Stitch Networks make use of cross-stitch units to connect two separate models together. Figure 12 below shows an illustration of the cross-stitch units. At each layer of the Cross-Stitch Network, a cross-stitch unit learns a linear combination of the activation maps from two different models. This allows sharing of representations from the two separate models.

![Figure 12 - Cross-stitch networks for two models suggested by Misra and Shrivastava [13]](image)

In this project, we adapted the concept of cross-stitch units into our models. More concretely, Equation 8a and Equation 8b below are applied to activation maps for each layer of Model 1 and Model 2 (See Figure 11). \( A^1 \) and \( A^2 \) represents an Activation Map of Model 1 and Model 2 respectively. \( W \) refers to the Weights of the Cross-Stitch Units. A linear combination of the Activation Maps of the two models is taken to produce a stitched activation map (\( A^1_s \) or \( A^2_s \)) in the cross stitch unit. We have one deviation from the cross-stitched network suggested in [13]. Misra and Shrivastava [13] fixed the 4 different \( W \) in Equation 8a and Equation 8b to be a fixed scalar constant for each layer. But in this paper, \( W \) is a matrix of the same shape as its corresponding \( A \). \( W \) contains a matrix of parameters which can be independently tuned. The \( \cdot \) operator represents element-wise multiplication. We made this deviation so as to introduce more flexibility in the cross-stitched network and hopefully achieve better results.

\[
A^1_s = A^1 \cdot W^1_{11} + A^2 \cdot W^2_{12} \quad \quad (8a)
\]

\[
A^2_s = A^1 \cdot W^1_{21} + A^2 \cdot W^2_{22} \quad \quad (8b)
\]

Note that within 1 cross-stitch unit, there are 2 different stitched activation maps (\( A^1_s \) and \( A^2_s \)); Where \( A^1_s \) represents the stitched activation map for Model 1; \( A^2_s \) represents the stitched activation map for Model 2; \( A^1 \) and \( A^2 \) represents the activation map of Model 1 and Model 2 respectively for one of their hidden layers. \( W^1_{11} \) and \( W^2_{12} \) represents the Weight matrices for
Model 1 to stitch $A^1$ and $A^2$ together. $W_{21}$ and $W_{22}$ represents the Weight matrices for Model 2 to stitch $A^1$ and $A^2$ together. Equation 8a and Equation 8b is applied to every layer of Models 1 and 2. In other words, for the kth layer of Models 1 and Model 2, there will be a corresponding kth cross-stitch unit.

The last stitched activation map for Model 1 (i.e. the last $A^1_*$ ) will be fed into a softmax layer in Model 1 to obtain the Emotion classification scores. Similarly, Model 2 will do the same with its last $A^2_*$ to obtain the Gender classification scores.

Referring to Table 16 below, we can see that with Cross-stitching, the test accuracy of Model 1 markedly improved after Cross-stitching is applied. This suggests the Model 1 is able to learn important information from the hidden activations provided by Model 2. On the other hand, while Model 2’s test accuracy did not improve, the Cross-stitching has not worsened the performance of Model 2 either. Model 1 and Model 2 use the same Dataset with the same Train-Test sets split, with differing labels (Emotion vs Gender). The Cross-stitching is only done for hidden layer 2 (last hidden layer) of the Models.

Table 16 - Results of Cross-stitching the Models together

<table>
<thead>
<tr>
<th></th>
<th>Model 1 (Emotion Classifier)</th>
<th>Model 2 (Gender Classifier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Cross-stitching, Test Accuracy</td>
<td>0.787</td>
<td>0.985</td>
</tr>
<tr>
<td>With Cross-stitching, Test Accuracy</td>
<td>0.801</td>
<td>0.985</td>
</tr>
</tbody>
</table>

4.1.2 Sluice Networks

Sluice Networks is a newer MTL approach proposed by Ruder [14]. It is a generalisation of Cross-stitch Networks. Like Cross-stitch Networks, the Sluice Networks also uses cross-stitched units to share representations between two models. Refer to Figure 13 below for an illustration of the cross-stitched units.
The key distinguishing feature of the Sluice Network from Cross-stitch Network is the red arrows as shown in Figure 13. The red arrows represent a set of weights $\beta$. The $\beta$ allows Model 1 (left of Figure 13) to make predictions by using all 2 Cross Stitch Units instead of just the deepest Cross Stitch Unit as we had seen in Cross-stitched Network in Section 4.1.1. For example, let $A_s^1$ and $A_s^2$ represent the stitched activation maps for Model 1 and Model 2 respectively in the first Cross-stitch unit. And let $B_s^1$ and $B_s^2$ represent the same in the second Cross-stitch unit. Then $\beta$ for Model 1 will have the same shape as the concatenation of $[A_s^1, B_s^1]$. The element-wise multiplication of $\beta$ with concatenation $[A_s^1, B_s^1]$ will return the Final Stitched Map for Model 1 in Figure 13. The same can be said for Model 2, but with a different $\beta$.

The intuition behind this is that the task performed by Model 1 may require not just high-level features in the deepest layer of the network but also low-level features in the shallower layers in order to maximise performance. The same can be said for Model 2. Hence, by allowing the Models to learn the parameter $\beta$, the Models can decide how much of the shared low-level features in the first cross-stitch unit and of the shared high-level features in the second cross-stitch unit to use in making the final predictions for their tasks.

Table 17 - Results of Sluice Network

<table>
<thead>
<tr>
<th></th>
<th>Model 1 (Emotion Classifier)</th>
<th>Model 2 (Gender Classifier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Sluice Network, Test Accuracy</td>
<td>0.787</td>
<td>0.985</td>
</tr>
<tr>
<td>With Sluice Network, Test Accuracy</td>
<td>0.801</td>
<td>0.985</td>
</tr>
</tbody>
</table>

Referring to Table 17 above, we can see that using Sluice Network improves Model 1’s performance and did not worsen the performance of Model 2. The improvement produced by Sluice Network is similar in terms of magnitude with that of the Cross-Stitched Network.
results. Perhaps a different approach for MTL is required. In the following sections, we describe and experiment with self-designed MTL techniques that is based on this idea: If we can make use of Model 2 (Gender Classifier) to remove gender information in the hidden representations of Model 1 (Emotion Classifier), perhaps there is a good chance that the performance of Model 1 can be improved.

### 4.1.3 Weight Suppression MTL

Other than depending on the Models to learn how they would like to share the different representations, we also conduct experiments on manually manipulating the Weights in each Model with the aim to share learning between the two Models. Referring to Figure 14 below, we manually alter $W^E_1$ based on the values in $W^G_1$. We refer to this novel technique as Weight Suppression.

We will first train Autoencoder layers (Both Hidden layer 1 and Hidden Layer 2) in both Model 1 and Model 2. Then, we will train Model 2 to do Gender Classification. Then, we train Model 1 to do Emotion Classification but after every training epoch, we will apply the algorithm in Figure 15 to suppress certain identified weights in $W^E_1$.

![Figure 14 - Diagram for Weight Suppression as a means to do MTL. The Values in $W^E_1$ are being suppressed based on values from $W^G_1$.](image)

*If element $i$ in $W^G_1$ is $< -0.35$ or $> 0.35$, then the corresponding element $j$ in $W^E_1$ will be multiplied by 0.99. *The threshold value of 0.35 is picked after inspecting the min, mean, median, and max of the elements in $W^E_1$. Note that $W^G_1$ and $W^E_1$ are required to have the same shape of [number_of_input_dimensions, dimension_of_hidden_layer_1].

![Figure 15 - Algorithm for Weight Suppression](image)

The intuition behind the algorithm is that when the Model 2 (Gender Classifier) believes that certain parts of the PRAAT Features are important for Gender Classification, we will reduce the effect or importance of those parts of PRAAT Feature in Model 1 (Emotion Classifier). This is based on the assumption that if a part of the PRAAT Feature is being heavily used for Gender classification, then it is less relevant for Emotion classification. In order to prevent the Emotion Classifier from using these identified parts of PRAAT Features, we simply suppressed the
elements in $W^E_1$ just as we had specified in the algorithm.

As seen in Table 18, the Weight Suppression MTL has led to a slight improvement in the Emotion Classification score of Model 1. We believe that although we have positive results, the improvement in test accuracy is too minor to be statistically significant.

Further experiments whereby we decrease the multiplicative factor from 0.99 to up till 0.80 yield similar results.

### 4.1.4 Input Suppression MTL

Input Suppression is a variant of Weight Suppression. As before, we train the Autoencoder layers in both Model 1 and Model 2. We then train Model 2 to do gender classification so as to obtain a trained $W^G_1$ Matrix (same as in Figure 14). Then, referring to Figure 16 below, we will first sum up the absolute values of each row of the $W^G_1$ Matrix to get a vector i.e. row-wise summation. We then identify the largest elements in the vector. These elements will indicate which row in the $W^G_1$ Matrix is important for Gender Classification. And since each row represents and corresponds to 1 PRAAT feature, by identifying the important rows, we essentially identified the important PRAAT features for Gender Classification. We require $W^G_1$ and $W^E_1$ to have the same shape. Thus each row in $W^G_1$ can be mapped to a corresponding row in $W^E_1$.

Then, we start to train Model 1 to do Emotion Classification. To ensure robust Emotion Classification that is independent of Gender information, at epoch 100, we will set the corresponding rows in the $W^E_1$ Matrix to 0, so as to diminish the use of PRAAT features that had been important for Gender classification. Training resumes as per normal until epoch 1000.

![Figure 16 - Breakdown of the $W^G_1$ Matrix](image)

Table 18 - Results of Weight Suppression

<table>
<thead>
<tr>
<th></th>
<th>Model 1 (Emotion Classifier)</th>
<th>Model 2 (Gender Classifier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Weight Suppression, Test Accuracy</td>
<td>0.787</td>
<td>0.985</td>
</tr>
<tr>
<td>With Weight Suppression, Test Accuracy</td>
<td>0.794</td>
<td>0.985</td>
</tr>
</tbody>
</table>
Table 19 - Results of Input Suppression

<table>
<thead>
<tr>
<th></th>
<th>Model 1 (Emotion Classifier)</th>
<th>Model 2 (Gender Classifier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Input Suppression, Test Accuracy</td>
<td>0.787</td>
<td>0.985</td>
</tr>
<tr>
<td>With Input Suppression, Test Accuracy</td>
<td>0.794</td>
<td>0.985</td>
</tr>
</tbody>
</table>

Referring to Table 19 above, similar to Weight Suppression, Input Suppression led to only slight improvement in the results, and that is not statistically significant.

### 4.1.5 Correlated Information Removal MTL

![Figure 17 - Modified Architecture of Model 1](image)

In order to carry out Correlated Information Removal MTL, we first modified the architecture of Model 1 (Emotion Classifier) by adding the Weight $W_{Sup}^G$ so as to allow Model 1 to learn to classify Gender Classes as well. The modified architecture is shown in Figure 17 above.

There is a total of three phases: 1. first training phase 2. second training phase 3. finetune training phase. During the first training phase, we train Model 1 to classify the Emotion Classes, as usual. After that, we conduct the second training phase which is to train Model 1 to learn to classify the Gender Classes. During the second training phase, the only weights that are trained or modified is $W_{Sup}^E$. The other weights are held constant.

After the second training phase, we seek to identify which are the hidden units in the Hidden
Layer 2 that are useful for identifying the Gender classes\(^\text{14}\). We then remove or diminish the importance or effect of these hidden units. Concretely, referring to Figure 18 below, once we had identified the hidden units (i.e. the rows in \( W_{\text{sup}}^G \)) which are important for identifying the Gender classes, we will set the corresponding rows in \( W_{\text{sup}}^E \) to 0. We then conduct finetune training for Model 1 again to re-learn how to recognise the Emotion Classes. During the finetune training, all weights of Model 1 are allowed to be changed. This helps Model 1 make predictions on Emotion classes in a way that is robust to the Gender information.

![Figure 18 - The workings of Correlated Information Removal MTL](image)

Table 20 - Results of Correlated Information Removal MTL

| Without Correlated Information Removal, Test Accuracy | 0.787 |
| With Correlated Information Removal, Test Accuracy | 0.794 |

Table 21 - Results of Correlated Information Removal MTL with Supervised Loss and Triplet Loss

| Without Correlated Information Removal, Test Accuracy | 0.838 |
| With Correlated Information Removal, Test Accuracy | 0.846 |

From Table 20 above, we can see that Correlated Information Removal MTL led to a modest increase in test accuracy. A possible explanation is that Model 1 is able to perform better as the deliberate setting of weights which are important to Gender Information to 0 helps Model 1 to

\[^\text{14}\text{ We take the row-wise summation of } W_{\text{sup}}^G \text{ to obtain a vector. The top few elements of the vector will indicate which rows are important for gender classification. Since each row of } W_{\text{sup}}^G \text{ corresponds to a hidden unit in Hidden Layer 2, we can able to identify which hidden unit is important for gender classification.} \]
move to a new optimal state where it is less sensitive and thus less disrupted by noisy gender information (contained in the input signals) that is irrelevant to emotion classification. However, we note that the improvement in results is still modest and is unlikely to be statistically significant.

In Table 21 above, we repeat the same experiment, but we included supervised loss and triplet loss in the training of Model 1. The reason why we do this is that we believe that Correlated Information Removal is trying to generate gender-invariant representations of input signals in Hidden Layer 2 of Model 1. So if we use other techniques that aid in obtaining gender-invariant representations (such as Supervised Loss and Triplet Loss), we should see a more pronounced improvement in the test accuracy than what we had seen in Table 20. Unfortunately, the resulting improvement in test accuracy in Table 21 is very similar in magnitude to what we have seen in the case without supervised loss and triple loss in Table 20.

In addition, we tried another experiment. Previously, for both the first and second training phases, we use a typical train-test sets split such that both training and test sets have instances of both male and female speakers. However, we conducted an additional experiment where during the first training phase, we fixed the female speakers to be in the train set, and the male speakers to be in the test set. Then, in the second training phase, we use back the regular train-test split that contains female and male instances in both train and test sets. The rationale is to let Model 1 be aware of only the instances from female speakers and let it train its weights in such a way that it makes full use of the information contained in the input signals, whether associated with gender or not, to predict the emotion classes. Then, in the second training phase, we try to identify the weights in Model 1 that are associated with gender information. We set much of those weights to zeros and then finetune/re-train Model 1 using the female-male train-test sets split, and see if Model 1 could capture a more robust, gender-invariant representation of the instances due to Correlated Information Removal. We report the results in Table 22 below.

Table 22 - Results of Correlated Information Removal MTL with Female-male Train-test split after the Finetuning phase

<table>
<thead>
<tr>
<th>Model 1 (Emotion Classifier) with Supervised Loss and Triplet Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Correlated Information Removal, Test Accuracy</td>
</tr>
<tr>
<td>0.532</td>
</tr>
<tr>
<td>With Correlated Information Removal, Test Accuracy</td>
</tr>
<tr>
<td>0.540</td>
</tr>
</tbody>
</table>

As shown in Table 22 above, the test accuracy of Model 1 improved after Correlated Information Removal, but only slightly. It appears that using Female-male Train-test split or not does not affect performance.
4.1.6 Gender Information Capitalization MTL

We start with the typical train-test sets split. Using the Training set, we further split it into two subsets. We place all the instances associated with female speakers into Subset 1, and we place all the male speakers instances into Subset 2. We will train Model 1 (Note that we use 2 different instances of Model 1 here) on each on these subsets separately to get 2 different last layer weights - $W_{Sup,1}^E$ and $W_{Sup,2}^E$. An illustration is provided in Figure 19. Note that both instances of Model 1 are emotion classifiers.

Figure 19 - Diagram of Gender Information Capitalization MTL

$$W_{Diff} = \left| W_{Sup,1}^E - W_{Sup,2}^E \right| \quad (9)$$

Where $W_{Sup,1}^E$ is the Weight matrix for the last layer of the 1st instance of Model 1, and $W_{Sup,2}^E$ is the Weight matrix for the last layer of the 2nd instance of Model 2. $W_{Diff}$ is the difference between the two Weight matrices.
We first compute the difference between $W_{\text{Sup}_1}^E$ and $W_{\text{Sup}_2}^E$ to get $W_{\text{Diff}}$ as specified in Equation 9. Referring to Figure 20, we find the largest values in $W_{\text{Diff}}$ and then identify the corresponding pair of elements in $W_{\text{Sup}_1}^E$ and $W_{\text{Sup}_2}^E$. We take the average of the pair of elements and insert the value back to the corresponding positions in $W_{\text{Sup}_1}^E$ and $W_{\text{Sup}_2}^E$. The intuition is that we want to find the difference between weights learned for Female Speakers and the weights learned for Male Speakers (i.e. $W_{\text{Diff}}$). We identify the weights that have the largest difference. These identified weights represent what is most different between a classifier trained on female speakers and a classifier trained on male speakers. We then identify the corresponding pairs of weights from $W_{\text{Sup}_1}^E$ and $W_{\text{Sup}_2}^E$, and then average the pair of weights to find the middle between the weight value for identifying emotions of Female speakers and the weight value for identifying emotions of Male speakers. We insert this average value back into both $W_{\text{Sup}_1}^E$ and $W_{\text{Sup}_2}^E$. This averaging process is a way of telling both instances of Model 1 that we want an emotion classifier that is capable of classifying both male and female instances equally well (i.e. a gender-invariant emotion classifier).

Table 23 - Results of Gender Information Capitalization MTL

<table>
<thead>
<tr>
<th></th>
<th>Model 1 (1st Instance) Test Acc</th>
<th>Model 1 (2nd Instance) Test Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>After training with their Subset</td>
<td>0.640</td>
<td>0.522</td>
</tr>
<tr>
<td>After Gender Information Capitalization MTL</td>
<td>0.654</td>
<td>0.544</td>
</tr>
<tr>
<td>Re-combine the 2 Subsets, and re-traine Model 1</td>
<td>0.809</td>
<td>0.787</td>
</tr>
<tr>
<td>Without Gender Information Capitalization MTL$^{15}$</td>
<td>0.787</td>
<td></td>
</tr>
</tbody>
</table>

$^{15}$Training is done using the combined 2 Subsets as the training set.
Referring to Table 23, we obtain test accuracies of 0.640 and 0.522 when we train the two instances of Model 1 with their respective Training Subsets, without using Gender Information Capitalization MTL. When we apply Gender Information Capitalization MTL, we observed that the test accuracies of the two Model 1 instances improved to 0.654 and 0.544 after modifying the \( W_{sup,1} \) and \( W_{sup,2} \) of the Model 1 instances and re-training them with their respective subsets (subsets not re-combined).

Furthermore, when we re-combine the two Training Subsets to get back the original training set and re-train the 2 instances of Model 1 with the training set, we obtain a test accuracy of 0.809 for 1st instance of Model 1. We consider only the 1st instance of Model 1 as it has a higher test accuracy than the 2nd instance of Model 1. The test score of 0.809 is a significant improvement from the score of 0.787 which is obtained when we do not use Gender Information Capitalization MTL at all. This suggests that Gender Information Capitalization MTL may be a useful MTL algorithm to use in order to obtain gender-invariant representations of the input signals.

### 4.2 Further Investigation into Correlated Information Removal MTL in Section 4.1.5

In Section 4.1.5, we notice that there is a curious and interesting observation in Correlated Information Removal MTL. As shown in Figure 21, we observe that when we modified the architecture of Model 1 (Emotion Classifier) in Correlated Information Removal MTL, the Gender Classification branch of Model 1 achieved a high test accuracy of 88.97% for gender classification. This is an unusual result because Model 1 is supposed to have eliminated or suppressed Gender Information via its two layers of Autoencoders (AE). But the test accuracy of 88.97% suggests that activations of Hidden Layer 2 still contain plenty of gender information. This is an interesting observation that we attempt to exploit later. It suggests the AE layers failed to sufficiently remove gender information from the PRAAT features and that the AE layers should be replaced with other techniques that could actually remove gender information adequately.

![Figure 21 - Illustration of Model 1 used with Correlated Information Removal MTL](image)
The issue with using Correlated Information Removal MTL to remove the gender information in the activation map of Hidden Layer 2 (see Figure 18 in Section 4.1.5) is that it does not guarantee that the Gender Information in Hidden Layer 2 will be removed. Correlated Information Removal MTL simply sets the relevant rows in $W_{sup}^E$ to 0 so that the Softmax Layer for Emotion Classification will not over-depend on the hidden units in Hidden Layer 2 that are important in Gender Classification (since each row in $W_{sup}^E$ correspond to a hidden unit in Hidden Layer 2). However, it is possible that those rows in $W_{sup}^E$ that had been set to 0 will re-learn back their original values during the finetuning training stage of Correlated Information Removal MTL. Therefore, other than using Correlated Information Removal MTL, we explore other possible techniques, namely 1.Joint Cost Optimization, 2.Noise Optimization, 3. Inhibited Learning in later sections.

### 4.2.1 Joint Cost Optimization

Both Joint Cost Optimization and Noise Optimization are variants of Correlated Information Removal MTL and we use the same modified Model 1 illustrated in Figure 21 above. After we modified Model 1 by adding the branch for Gender Classification, we will first train Model 1 for emotion classification. Then we train its Gender Classification branch by learning only $W_{sup}^G$ while holding the other weights in Model 1 constant. After this, we commence Joint Cost Optimization by minimising the Cost Function $J_{Joint}$ in Equation 10 below.

$$J_{Joint} = J_{Emo} - \alpha \cdot J_{Gen} \quad (10)$$

Where $J_{Joint}$ is the overall joint loss for training; $J_{Emo}$ is Softmax Cost for Emotion Classification in Model 1; $J_{Gen}$ is Softmax Cost for Gender Classification in Model 1, and $\alpha$ is an arbitrary weighting factor.

When we are training $J_{Joint}$, we will only modify or learn the weights of $W_{sup}^E$, $W_1^E$ and $W_2^E$ (And their corresponding bias units i.e. $b_{sup}^E$, $b_1^E$ and $b_2^E$). Intuitively, we are asking Model 1 to re-learn how it represents the input instances in Hidden Layer 2 such that the ability for the activations of Hidden Layer 2 to be used for Gender Classification is minimized (since we seek to minimize “$- \alpha J_{Gen}$” in Equation 10). We included “$J_{Emo}$” in “$J_{Joint}$” so that the newly learned activations of Hidden Layer 2 will remain useful for Emotion Classification. After training $J_{Joint}$, We will again re-train $W_{sup}^G$ as in the previous step so as to check whether the newly learned hidden representation in Hidden Layer 2 still contains a lot of gender information or not. If there is still a lot of gender information, then we need to train $J_{Joint}$ again. A summary of the algorithm of Joint Cost Optimization is provided in Figure 22 below. Note that $T$ is an arbitrary threshold.

---

16 The finetuning training stage the final phase of Correlated Information Removal MTL. It is a round of training carried out for the modified Model 1 after we have set the $W_{sup}^E$ to 0.
1: Minimize $J_{Gen}$ by learning only $W_{Sup}^G$ [All other weights are held constant]
2: If Test Accuracy for Gender Classification is $>\$Threshold T after step 1, go to step 3. Otherwise terminate algorithm.
3: Minimize $J_{Joint} = J_{Emo} - \alpha J_{Gen}$ by learning only $\{W_{Sup}^{E}, W_1^{E}, W_2^{E}, b_{Sup}^{E}, b_1^{E} \text{ and } b_2^{E}\}$. Go back to step 1.

Figure 22 - Algorithm for Joint Cost Optimization

Table 24 - Results after 1 iteration of Joint Cost Optimization

<table>
<thead>
<tr>
<th>Step no. based on Algorithm in Figure 22</th>
<th>Model 1’s Gender Classification Test Acc (We want to Minimise this)</th>
<th>Model 1’s Emotion Classification Test Acc (We want to Maximise this)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Step 1</td>
<td>NA</td>
<td>0.801</td>
</tr>
<tr>
<td>After Step 1</td>
<td>0.890</td>
<td>0.801$^{17}$</td>
</tr>
<tr>
<td>After Steps 2 &amp; 3</td>
<td>NA$^{18}$</td>
<td>0.809</td>
</tr>
<tr>
<td>After Step 1 again</td>
<td>0.699</td>
<td>0.809$^{19}$</td>
</tr>
</tbody>
</table>

Referring to Table 24, we can see that Joint Cost Optimization successfully reduced the gender test accuracy from 0.890 to 0.699, thereby indicating that the Gender information from the activations of Hidden Layer 2 in Model 1 has been removed. While there has also been a slight improvement in the Emotion Classification Test accuracy after removing the Gender information, the improvement is not significant. Further iterations failed to improve Emotion Classification Test accuracy. Similarly, further experiments with different parameter values for threshold $T$ and weighing factor $\alpha$ failed to significantly improve the Emotion Classification Test accuracy.

### 4.2.2 Noise Optimization

Noise Optimization is a modified version of Joint Cost Optimization whereby we changed the Step 3 of Joint Cost Optimization into two Steps: 1. Minimize the negative $J_{Gen}$ by changing the weights of just $W_1^{E}, W_2^{E}, b_1^{E} \text{ and } b_2^{E}$. 2. Then, Minimize $J_{Emo}$ by learning only $W_{Sup}^{E}, W_1^{E}, W_2^{E}, b_{Sup}^{E}, b_1^{E} \text{ and } b_2^{E}$. The intuition behind minimising negative $J_{Gen}$ is to introduce “noise” to the Hidden Layer 2 activations such that they will no longer contain much gender information. Then, we will re-learn how to classify the Hidden Layer 2 activations into emotion classes by minimizing $J_{Emo}$.

---

$^{17}$ Unchanged since Step 1 will not affect Emotion Classification in Model 1

$^{18}$ Because Hidden Layer 2 Activations have changed, thus Gender Classification score will also change. To find the new Gender Classification score, we need to repeat Step 1

$^{19}$ Step 1 will not affect Emotion Classification
1: Minimize $J_{\text{gen}}$ by learning only $W_{\text{sup}}^G$. [All other weights are held constant]
2: If Test Accuracy for Gender Classification is $>$ Threshold $T$ after step 1, go to step 3. Otherwise terminate Algorithm
3: Minimize $-J_{\text{gen}}$ by learning only $\{W_1^E, W_2^E, b_1^E, b_2^E\}$
4: Minimize $J_{\text{ent}}$ by learning only $\{W_{\text{sup}}^E, W_1^E, W_2^E, b_{\text{sup}}^E, b_1^E, b_2^E\}$. Go Back to step 1.

Figure 23 - Algorithm for Noise Optimization

<table>
<thead>
<tr>
<th>Step no. based on Algorithm in Figure 23</th>
<th>Model 1’s Gender Classification Test Accuracy (We want to Minimise this)</th>
<th>Model 1’s Emotion Classification Test Accuracy (We want to Maximise this)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Step 1</td>
<td>NA</td>
<td>0.801</td>
</tr>
<tr>
<td>After Step 1</td>
<td>0.890</td>
<td>0.801\textsuperscript{20}</td>
</tr>
<tr>
<td>After Step 2</td>
<td>NA\textsuperscript{21}</td>
<td>0.824</td>
</tr>
<tr>
<td>After Step 1 again</td>
<td>0.846</td>
<td>0.824\textsuperscript{22}</td>
</tr>
<tr>
<td>After Step 2 again</td>
<td>NA</td>
<td>0.801</td>
</tr>
<tr>
<td>After Step 1 again</td>
<td>0.772</td>
<td>0.801</td>
</tr>
</tbody>
</table>

Referring to Table 25, we notice that after 1 iteration of Noise Optimization, the Gender Classification Test Accuracy drops slightly from 0.890 to 0.846, and the Emotion Classification Test Accuracy increased from 0.801 to 0.824. While 0.824 shows a marked improvement in Emotion classification, the Gender Classification Test Accuracy did not fall by much, suggesting that gender information in Hidden Layer 2 activations still remains significant. In Iteration 2, the Gender Classification Test Accuracy drops to 0.772, but the Emotion Classification Test Accuracy also falls to 0.801.

The results are actually very interesting. It appears to indicate that without Gender Information, the Emotion Classification score will suffer as well. We learn to accept and exploit this in a later experiment called ‘Stacked Model’ written in Section 4.3.2.

In addition, we conduct experiments that combine both Joint Cost Optimization and Noise Optimization, the results are similar to their individual performances and also failed to outperform Correlated Information Removal MTL.

\textsuperscript{20} Unchanged since Step 1 will not affect Emotion Classification in Model 1
\textsuperscript{21} Because Hidden Layer 2 Activations have changed, thus Gender Classification score will also change. To find the new Gender Classification score, we need to repeat Step 1
\textsuperscript{22} Step 1 will not affect Emotion Classification
4.2.3 Inhibited Learning

In this section, we present another variant of Correlated Information Removal MTL. Refer back to Figure 18 in Correlated Information Removal MTL. For Correlated Information Removal, in order to identify which of the rows of Matrix $W_{Sup}^G$ (and hence which corresponding hidden units in Hidden Layer 2) are important for gender classification, we first sum up elements of each row in the Matrix to obtain a Vector (row-wise summation). We then get the top few elements in the Vector. This operation is also depicted in (a) of Figure 25 below. However, taking the row-wise sum may not be an accurate estimation of how important a hidden unit is to the Gender Classification task. Rather than taking the row-wise sum of $W_{Sup}^G$, we should actually compute the absolute value of the row-wise subtraction. We explain the intuition behind this in Figure 24 below. Whether $H_0$ Neuron is important in Gender Classification is not so much dependent on the Sum of $w_{00}$ and $w_{01}$ (i.e. if we take row-wise summation) as compared to magnitude of the Difference between $w_{00}$ and $w_{01}$.

![Figure 24 - Intuition behind taking the Differences between weights rather than the Sum.](image)

![Figure 25 - Comparison of two different ways - (a) and (b) - to identify the important Hidden Neurons in Hidden Layer 2.](image)
In (b) in Figure 25 above, we illustrate how we identify the most important hidden units in Hidden Layer using $W_{\text{sup}}^{E}$ in our Inhibited Learning technique. We use a model architecture as depicted in Figure 26 below. After training $W_{\text{sup}}^{E}$ and $W_{1}^{E}$ (keeping $W_{\text{sup}}^{G}$ fixed) to predict Emotion Classes, we will train $W_{\text{sup}}^{G}$ (keep other weights fixed) to predict Gender Classes. Then, we will identify the important Hidden Units in $W_{\text{sup}}^{G}$, similar to what has been depicted in (b) of Figure 25. These identified hidden units tell us which weights in $W_{\text{sup}}^{E}$ are important for Gender Classification (see Figure 18 in Section 4.1.5). After that, we will re-initialize all the values in $W_{\text{sup}}^{E}$ and $W_{1}^{E}$. We proceed with re-training of $W_{\text{sup}}^{E}$ and $W_{1}^{E}$ (keeping $W_{\text{sup}}^{G}$ fixed) to predict Emotion Classes. During the re-training, we inhibit the learning of the weights in $W_{\text{sup}}^{E}$ which are important for Gender Classification. We inhibit the learning of these identified weights by reducing the learning rate of these weights (learning rates of the other weights are not affected). We summarize the Inhibited Learning Algorithm in Figure 27 below.

![Diagram](image.png)

Figure 26 - Model Architecture used in Inhibited Learning

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Minimize $J_{\text{Emotion}}$ (Softmax Loss for Emotion Classification) by learning only { $W_{\text{sup}}^{E}$, $W_{1}^{E}$ }</td>
</tr>
<tr>
<td>2</td>
<td>Minimize $J_{\text{Gender}}$ (Softmax Loss for Gender Classification) by learning only $W_{\text{sup}}^{G}$ [All other weights are held constant]</td>
</tr>
<tr>
<td>3</td>
<td>Identify Weights in $W_{\text{sup}}^{E}$ that are important for Gender Classification</td>
</tr>
<tr>
<td>4</td>
<td>Re-initialize and Re-train only { $W_{\text{sup}}^{E}$, $W_{1}^{E}$ } to Minimize $J_{\text{Emotion}}$. During Re-training, inhibit learning of the identified important weights in $W_{\text{sup}}^{E}$ by reducing their learning rates [other weights’ in $W_{\text{sup}}^{E}$ learning rate are not affected]</td>
</tr>
</tbody>
</table>

Figure 27 - Algorithm for Inhibited Learning

<table>
<thead>
<tr>
<th>Emotion Test Acc using Imbalanced Gender Train-Test Sets split 23</th>
<th>Fully-Connected Network</th>
<th>Network with Inhibited Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.727</td>
<td>0.740</td>
</tr>
</tbody>
</table>

We present the results of Inhibited Learning in Table 26. Note that both models in Table 26 only has 1 Hidden Layer each. We use an Imbalanced Gender Train-Test Sets split 2, which makes it more important that our model removes gender information in the training set. We observe that

23 Imbalanced Gender Train-Test Sets split refers to having a much higher female-to-male speakers ratio in Train set as compared to Test set. More details will be given in Section 4.2.5 later.
Inhibited Learning improves the test accuracy compared to just using a fully-connected neural network with 1 hidden later. Our experiments with normal train-test splits of the dataset seems to achieve negative results, indicating that inhibited learning does not improve test accuracy when the train set and test set has similar or same ratio of female-to-male speakers.

### 4.2.4 Exploiting Observations from Correlated Information Removal MTL to design a Replacement of Autoencoders

One of the reasons that we used Autoencoders (AE) in our experiments is that we hope that it extracts only the most essential information from the input signals. Ideally, the representations from the AE should be robust to gender differences between the speakers. In other words, we hope that the AE representations do not contain any (or too much) gender information. This, however, is a naive assumption because in Section 4.2 (Gender Classification branch of Figure 21 achieving a high test accuracy of 88.97%), we observed that the AE representations, in fact, contains plenty of gender information. We hypothesize that removing the gender information from the representations will make it easier for the model to make predictions on the emotion classes because Gender information in representations will, to some extent, act as noise when we are trying to predict emotions.

Since AE is clearly not doing a good job of removing gender information, we propose to replace AE with an algorithm based on a slight adaptation of the Joint Cost Optimization (described in Section 4.2.1). This algorithm is similar to what we described in Section 4.2.1 except that we are applying it in Hidden layer 1 here instead of Hidden layer 2. Thus, we will just call this algorithm the Joint Cost Optimization algorithm. We will describe more details on it later.

![Comparison of Model Architecture](image)

Figure 28 - Comparison of the Model Architecture when using AE and using Joint Cost Optimization algorithm. Note that the bias units are omitted from this figure.

To investigate the improvement, if any, of replacing AE with Joint Cost Optimization algorithm, we set two different models - Model 1 and Model 2 as depicted on the left and right of Figure 28 respectively. Model 1 is a simple single layer AE where the encoded layer is used to predict Emotion Classes.

In contrast, Model 2 does not have an AE layer. In place of the AE layer is a fully connected

---

24 i.e. Replacing Autoencoders with Joint Cost Optimization Algorithm
hidden layer that seeks to optimize the prediction of Emotion Classes. Model 2 is first being trained to predict Emotion Classes where all weights are allowed to be changed during this training. After this, we begin training Model 2 to optimize Gender Classes where all weights except $W^G_{sup}$ are held fixed. After that, we re-train $W^E_f$ and $W^E_{sup}$ and their corresponding bias units by optimizing a joint cost function consisting of a weighted sum between the Softmax Cost Function for Emotion Classification and the Negative of the Softmax Cost Function for Gender Classification. During training of this joint cost function, the $W^G_{sup}$ is held fixed. We then re-train $W^G_{sup}$ to check the Gender Class Classification score again in order to determine if the gender information has been reduced in the Hidden layer representations.

Table 27 - Comparison of AE and Joint Cost Optimization Algorithm

<table>
<thead>
<tr>
<th></th>
<th>With AE (Model 1)</th>
<th>AE replaced by Joint Cost Optimization Algorithm(Model 2)</th>
<th>Fully Connected Network$^{25}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Before retraining$^{26}$ After retraining$^{27}$</td>
<td></td>
</tr>
<tr>
<td>Gender Classification Test Acc</td>
<td>NA</td>
<td>0.949</td>
<td>0.765</td>
</tr>
<tr>
<td>Emotion Classification Test Acc</td>
<td>0.728</td>
<td>0.824</td>
<td>0.801</td>
</tr>
</tbody>
</table>

Referring to Table 27, we observe that replacing AE with Joint Cost Optimization algorithm is able to bring about improvement in Emotion Class Classification from 0.728 to 0.824. Note that the Joint Cost Optimization algorithm successfully reduced the gender classification test accuracy from 0.949 to 0.765. This indicates that some gender information has been successfully removed in the Hidden Layer activations. Furthermore, we compare the Joint Cost Optimization algorithm with a Fully connected Network with 1 Hidden Layer (i.e. network with no AE at all). The Emotion Classification Test Accuracy of Joint Cost Optimization Algorithm remains superior. This suggests that removing the Gender information from hidden layer activations may actually improve Emotion Class Classification. This supports our hypothesis that the gender information in hidden representations will, to a certain extent, act as noise when we are trying to predict emotions.

$^{25}$ is essentially Model 2 without any Joint Cost Optimization or Gender Classification.
$^{26}$ Before re-training $W^E_f$ and $W^E_{sup}$
$^{27}$ After re-training $W^E_f$ and $W^E_{sup}$
4.2.5 Investigating Application of Joint Cost Optimization Algorithm on Imbalanced Gender Train-Test Sets split

The Joint Cost Optimization Algorithm used in Section 4.2.4 to replace Autoencoders is based on the intuition that the gender information in input instances acts as noise when we are predicting emotions.

Suppose a scenario where the Train Set contains a large proportion of Female speakers and a small proportion of Male Speakers. Concurrently, the Test Set contains a large proportion of Male speakers and a small proportion of Female Speakers. This is a difficult scenario for a Neural Net Model to deal with as the Train Set is significantly different from the Test Set. We will refer to this scenario as **Imbalanced Gender Train-Test Sets split** hereafter. Figure 29 below illustrates an Imbalanced Gender Train-Test Sets split using the EmoDB dataset.

![Figure 29 - Illustration of a Normal Train-Test Sets split and an Imbalanced Gender Train-Test Sets split using EmoDB dataset](image)

Previously in Section 4.2.4, we have seen how a Joint Cost Optimization outperformed an Autoencoder and Fully Connect Network in a normal Train-Test Sets split. We hypothesize that Joint Cost Optimization algorithm will work even better compared to Autoencoders or a Fully Connected Network in an Imbalanced Gender Train-Test Sets split scenario because Joint Cost Optimization algorithm is designed to remove gender information from its Hidden Layer activations. If the algorithm successfully removes all gender information from its Hidden Layer activations, then the Imbalanced Gender Train-Test Sets split problem should reduce into a normal Train-Test split. In other words, the algorithm should not perform any worse when presented with an Imbalanced Gender Train-Test Sets split as compared to its performance in a normal Train-Test split if it managed to remove all the gender information.

In this section, we will repeat the experiment in Table 27 of Section 4.2.4, but with an Imbalanced Gender Train-Test Sets split. We present the results in Table 28 below.
Table 28 - Comparison of AE and Joint Cost Optimization algorithm, using Imbalanced Gender Train-Test Sets split

|                      | With AE (Model 1) | AE replaced by Joint Cost Optimization Algorithm(Model 2) | Fully Connected Network  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before retraining</td>
<td>After retraining</td>
<td></td>
</tr>
<tr>
<td>Gender Test Acc</td>
<td>NA</td>
<td>0.935</td>
<td>0.734</td>
</tr>
<tr>
<td>Emotion Test Acc</td>
<td>0.666</td>
<td>0.747</td>
<td>0.727</td>
</tr>
</tbody>
</table>

Referring to Table 28, we observe that, similar to Table 27, the Joint Cost Optimization algorithm is able to outperform both AE and Fully connected network. However, Joint Cost Optimization is clearly affected by the Imbalanced Gender Train-Test Sets split and is unable to replicate the test accuracy of 0.824 as observed in Table 27 (where we have a normal Train-Test Sets split). Furthermore, Joint Cost Optimization is unable to increase its lead over Fully Connected network and AE. These observations may suggest that when the Joint Cost Optimization is trying to remove gender information from the Hidden activations, it has inadvertently introduced random noises to the Hidden activations.

4.3 Other Techniques that capitalise on Gender information in Training Data

As we had concluded at the end of ‘Section 4.2.2 Noise Optimization’, there seems to be evidence that gender information itself is actually useful or essential for emotion classification. In this section, we explore techniques that we could employ to improve the Emotion Classification of our models given that we have Gender information of the instances in the Training Set. We will explore two approaches - 1. Female-Male Convergence 2. Stacked Model

4.3.1 Female-Male Convergence

The idea behind Female-Male Convergence is that we will first train a model that take in instances of male speakers and instances of female speakers that belong to the same Emotion class (Using instances in the training set only). The model has a set of weights ($W_I$) which map these male and female instances into hidden representations. The model will seek to minimise the Mean Square Error (MSE) between the hidden representations of the male speakers and those of the female speakers belonging to the same Emotion class. Figure 30 shows the architecture of this model. As there are 7 Emotion Classes in EmoDB, we will update the weights of the model in 7

---

28 is essentially Model 2 without any Joint Cost Optimization or Gender Classification.
29 Before re-training $W_I^E$ and $W_{sup}^E$
30 After re-training $W_I^E$ and $W_{sup}^E$
steps. The first step will compute the MSE between the hidden representations of the male and female speakers of the first emotion class. The second step will do the same for the second emotion class, and so on for the other 5 steps.

![Figure 30 - Model Architecture for Female-Male Convergence](image)

However, there is an issue with this set-up. The current set-up is simply telling \( W_1 \) to map a Female and a Male instance belonging to the same Emotion class into two hidden representations that are similar to each other. The model can simply learn to map all instances into a zero-vector or some arbitrary noise vector. A way to circumvent this is to introduce \( W_{\text{prime}_1\text{female}} \) and \( W_{\text{prime}_1\text{male}} \) into the model. These two weights will map the learned Hidden Representations of female instances and male instances back to the original female and male PRAAT features respectively. We add the MSE between these reconstructed PRAAT features and the original PRAAT features as another component to the Cost Function of the model. This means that our Cost Function will take the form of Equation 11 below.

\[
J = MSE_1 + 0.5 \cdot MSE_{\text{Female}} + 0.5 \cdot MSE_{\text{Male}}
\]  

(11)

where \( MSE_1 \) is mean-squared error between the hidden activations of male and female speakers of the same emotion class; \( MSE_{\text{Female}} \) is the mean-squared error between the reconstructed PRAAT features and the original PRAAT features of female speakers. \( MSE_{\text{Male}} \) is the mean-squared error between the reconstructed PRAAT features and the original PRAAT features of male speakers.

After we learn the hidden representations for the speakers, we convert the entire train set and test set into hidden representations using our trained model. Then we build a new classifier that uses these new train and test sets. The classifier used in our experiments is a Fully connected Neural network with 1 hidden layer. We use Autoencoders for this hidden layer. We present our results in Table 29 below.

<table>
<thead>
<tr>
<th>Table 29 - Results of Female-Male Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emotion Test Accuracy</td>
</tr>
<tr>
<td>------------------------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

As seen in Table 29, we fail to obtain positive results using Female-Male Convergence. A possible reason is that the PRAAT features are too complicated such that it is impossible for male and female instances of 7 different emotion classes to be mapped to 7 common hidden representations using just \( W_1 \).
This is actually an interesting observation as the results suggest that mapping male and female instances into a common representation could prove too complicated. Thus, it kept us thinking on whether there is a simpler way to make use of the gender information that we had on the training instances. This leads directly to the next approach - Stacked Model

4.3.2 Stacked Model

The intuition is that since it is difficult to manually and directly combine male and female speakers into one common representation (as concluded from Section 4.3.1), it may be better to simply treat them as distinct. Here, we first train a gender classifier (Model 1) that predicts whether an instance is male or female. Then, we train two different models where one model (Model 2) will train on instances spoken by female speakers in training set, and the other model (Model 3) will train on instances spoken by male speakers. The architecture of the models is depicted in Figure 31 below.

After we trained all three models, Model 1 will be used to predict if an instance in test set is a male or female speaker. Then, based on the gender prediction, we will determine whether to send that instance to Model 2 (female speakers) or Model 3 (male speakers). All models have 0 hidden layers. This overall set-up of the models will be known as the Stacked Model. We present the results in Table 30.

![Figure 31 - Model Architecture for Stacked Model (Consists of 3 Models)](image-url)
Table 30 - Results of Stacked Model

<table>
<thead>
<tr>
<th></th>
<th>Test Acc of Model 2 (Female Speakers)</th>
<th>Test Acc of Model 3 (Male Speakers)</th>
<th>Combined Test Acc of Model 2 and Model 3</th>
<th>Fully Connected NN(^{31})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imbalanced Gender Train-Test Set</td>
<td>0.801</td>
<td>0.709</td>
<td>0.75</td>
<td>0.731</td>
</tr>
<tr>
<td>Balanced Gender Train-Test Sets</td>
<td>0.835</td>
<td>0.772</td>
<td>0.809</td>
<td>0.787</td>
</tr>
</tbody>
</table>

Referring to Table 30 above, we observe that the Stacked Model actually outperforms our control case (Fully Connected Neural Network with 0 Hidden Layer) in both Imbalanced Gender Train-Test Sets split\(^{32}\) and Balanced Train-Test Sets split. One notable observation is that the test accuracy of 0.75 in an Imbalanced Gender Train-Test Sets split is higher than those achieved by any other techniques or models explored by this paper. This may indicate that in an Imbalanced Gender Train-Test Sets split, the best approach may be to first apply a Gender Classifier before using an Emotion Classifier.

4.3.2.1 Class-based and Re-centered (CAR) Data Augmentation\(^{33}\)

From Table 30, we observed that Model 2 (Female Speakers) perform significantly better than Model 3 (Male Speakers). Now, we explore techniques to increase the performance of Model 3 (Male Speakers). We explore 2 techniques that attempt to do this - 1. Class-based and Re-centered (CAR) Data Augmentation 2. Selective Accelerated Learning (SAL). In this Section, we will briefly mention CAR’s results. Please read the footnotes on this page.

Table 31 - Results of using and not using CAR for Model 3

<table>
<thead>
<tr>
<th></th>
<th>Model 3 without CAR</th>
<th>Model 3 with CAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imbalanced Gender Train-Test Sets, Test Acc</td>
<td>0.709</td>
<td>0.744</td>
</tr>
</tbody>
</table>

In Table 31, we present results obtained from using and not using CAR. Clearly, the test accuracy of Model 3 has increased markedly after using CAR.

---

\(^{31}\) Neural Network with 0 Hidden Layer

\(^{32}\) An Imbalanced Train-Test split and a Normal (Balanced) Train-Test split had been described in Section 4.2.5

\(^{33}\) Due to a Technical Disclosure submission and a possible Patent application that centered around CAR and SAL, we are unable to provide more details on the two techniques.
4.3.2.2 Selective Accelerated Learning (SAL)

The second technique to increase the performance of Model 3 (Male Classifier) is called Selective Accelerated Learning (SAL). Again, due to a Technical Disclosure submission and a possible Patent application that centered around CAR and SAL, we are unable to provide more details on SAL.

As seen in Table 32, SAL helps to markedly improve the Test Accuracy of Model 3. However, SAL failed to improve the Test Accuracy of Model 2 as seen in Table 33.

| Table 32 - Results of using Selective Accelerated Learning on Male Model (Model 3) |
|-----------------------------------------------|-----------------|
| Model 3 w/o SAL | Model 3 with SAL |
| Imbalanced Gender Train-Test Sets, Test Acc | 0.709 | 0.738 |

| Table 33 - Results of using Selective Accelerated Learning on Female Model (Model 2) |
|-----------------------------------------------|-----------------|
| Model 2 w/o SAL | Model 2 w SAL |
| Balanced Gender Train-Test Sets, Test Acc | 0.835 | 0.833 |
| Imbalanced Gender Train-Test Sets, Test Acc | 0.801 | 0.780 |

| Table 34 - Results of using a Combination of CAR and SAL |
|-----------------------------------------------|-----------------|
| Model 3 w CAR | Model 3 with SAL | CAR + SAL |
| Imbalanced Gender Train-Test Sets, Test Acc | 0.744 | 0.738 | 0.744 |

In addition, we conducted experiments which investigate whether using both CAR and SAL together would obtain a better performance from Model 3 (Male Classifier). The results are presented in Table 34. The results indicate that using the 2 techniques concurrently would not improve the performance of Model 3 beyond what has been attained using just CAR. However, later we will show that combination of the 2 techniques can still lead to better results.

34 Model 3 is Model for Male Speakers
35 Model 2 is for Female Speakers
36 Again, Model 3 is for Male Speakers
4.4 Combining Datasets of Speeches vocalized in Different Languages

In this Section, we explore whether and how we could combine Datasets of speeches vocalized in different languages. Specifically, we look at combining the RAVDESS and EMODB Datasets. EMODB Dataset has 7 emotion classes and RAVDESS Dataset has 8 emotion classes. There are 6 common emotion classes between these two datasets. We will only look at training and test instances which belong to these 6 emotion classes in this Section.

The reason why we want to combine the two datasets is to explore whether CAR and SAL could be used to merge the information in RAVDESS and EMODB Datasets so as to improve a classifier performance in either RAVDESS or EMODB. We will make use of the CAR technique and the SAL technique.

Similar to Section 4.3.2, we will use two neural network classifiers. We will train and test the first neural network classifier on EMODB. Then, we will train and test the second neural network classifier on RAVDESS. Note that the input features used in the two models are the PRAAT features extracted from RAVDESS and EMODB.

4.4.1 Applying CAR technique on Datasets of different languages

The two neural network classifiers are very simple, fully connected neural network models with no hidden layers. The model architectures are illustrated in Figure 36. Note that we are only using instances belonging to the 6 emotion classes for both EMODB and RAVDESS.

Figure 36 - Models used for classifying the EMODB and RAVDESS Datasets
Table 35 - Results of Applying CAR technique on Datasets of different languages

<table>
<thead>
<tr>
<th>Model</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1 (EMODB classifier)</td>
<td>0.868</td>
</tr>
<tr>
<td>Model 2 (RAVDESS classifier)</td>
<td>0.802</td>
</tr>
<tr>
<td>Model 2 (RAVDESS classifier) with CAR</td>
<td>0.810</td>
</tr>
</tbody>
</table>

With CAR, we are able to improve Model 2 test accuracy from 0.802 to 0.810.

Previously, we have 460 train instances and 116 test instances in RAVDESS. Now, in order to constrain the training set of Model 2 and increase the importance of using CAR, we will use only 200 RAVDESS train instances for training and retain the same 116 RAVDESS test instances for testing. We will now refer to this new Training Set as the reduced Training Set hereafter.

Using the reduced Training Set, we repeat the experiments conducted for Table 35, and we present the results in Table 36 below.

Table 36 - Results of Applying CAR technique on Datasets of different languages on reduced Training Set

<table>
<thead>
<tr>
<th></th>
<th>Model 1 with EMODB</th>
<th>Model 2</th>
<th>Model 2</th>
<th>RAVDESS</th>
<th>RAVDESS + CAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Acc w/o excluded RAVDESS instances</td>
<td>0.868</td>
<td>0.75</td>
<td>0.767</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Acc w excluded RAVDESS instances</td>
<td>0.868</td>
<td>0.723</td>
<td>0.734</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 36 shows that CAR continues to lead to an improvement in the performance of Model 2. Comparing Table 35 and Table 36, we observe that by using a reduced Training Set for Model 2, the effect of using CAR technique is accentuated, as expected.
4.4.2 Applying SAL technique on Datasets of different languages

We use the same models as illustrated in Figure 36 and we apply SAL. We present our results in Table 37.

<table>
<thead>
<tr>
<th>Table 37 - Results of Applying SAL technique on Datasets of different languages for Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 2 Test Accuracy</td>
</tr>
<tr>
<td>Trained with RAVDESS w/o SAL</td>
</tr>
<tr>
<td>w SAL</td>
</tr>
<tr>
<td>w reduced Training Set, w/o excluded RAVDESS instances</td>
</tr>
<tr>
<td>w SAL, w reduced Training Set, w/o excluded RAVDESS instances</td>
</tr>
<tr>
<td>w reduced Train Set for Model 2, w excluded RAVDESS instances</td>
</tr>
<tr>
<td>w SAL, reduced Train Set, w excluded RAVDESS instances</td>
</tr>
</tbody>
</table>

In Table 37, we compare the difference between using and not using SAL. SAL improved the test accuracy of Model 2 from 0.802 to 0.810.

We also tried with the reduced Training Set for Model 2. With SAL, Model 2’s test accuracy improved from 0.75 to 0.784. Note that the 200 training instances that have been taken out of Model 2’s Train Set (as we are using the reduced Training Set) are not being placed into the Test Set of Model 2 here.

Then, we tried with the reduced Training Set and we also put the 200 training instances taken out of the Train Set into the Test Set of Model 2. With SAL, the test accuracy of Model 2 improved from 0.723 to 0.774.

---

37 Test Accuracy based on the 6 Common Classes between EMODB and RAVDESS
38 Without adding the RAVDESS train instances excluded from the reduced Training Set into Model 2’s Test Set
4.4.3 Combination of CAR and SAL on Datasets of different languages

We also experimented with using both CAR and SAL concurrently. We report the results in Table 38 below.

Table 38 - Results of using both CAR and SAL techniques on Datasets of different languages

<table>
<thead>
<tr>
<th>Model 2 Test Acc$^{39}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>w RAVDESS Train set</td>
</tr>
<tr>
<td>w CAR</td>
</tr>
<tr>
<td>w SAL</td>
</tr>
<tr>
<td>w CAR &amp; SAL</td>
</tr>
</tbody>
</table>

| w reduced Train Set, w/o excluded RAVDESS instances$^{40}$ | 0.75   |
| w reduced Train Set, w/o excluded RAVDESS instances + CAR | 0.767  |
| w reduced Train Set, w/o excluded RAVDESS instances + SAL | 0.784  |
| w reduced Train Set, w/o excluded RAVDESS instances + CAR & SAL | 0.793 |

| w reduced Train Set, w excluded RAVDESS instances$^{41}$ | 0.723 |
| w reduced Train Set, w excluded RAVDESS instances + CAR | 0.747 |
| w reduced Train Set, w excluded RAVDESS instances + SAL | 0.774 |
| w reduced Train Set, w excluded RAVDESS instances + CAR & SAL | 0.777 |

From Table 38, we observe that without using the reduced Training Set for Model 2 (using just a normal Training Set), using a combination of CAR and SAL techniques will not improve Model 2 beyond what the 2 techniques can do individually.

---

$^{39}$ Test Accuracy based on the 6 Common Classes between EMODB and RAVDESS

$^{40}$ Without adding the RAVDESS train instances excluded from the reduced Training Set into Model 2’s Test Set

$^{41}$ With adding the excluded RAVDESS train instances into Model 2’s Test Set
However, if we use the reduced Training Set for Model 2, regardless of whether we include the excluded RAVDESS instances or not, we observe that using both CAR and SAL concurrently can result in an improvement in the test accuracy of Model 2 beyond what CAR and SAL can achieve individually. This is evidence that CAR and SAL can still work together in some circumstances.

4.5 Performance of Selective Accelerated Learning in Image Domain

The improvements achieved by SAL in Section 4.3.2.2 and Section 4.4.2 provide evidence that SAL can help to improve test accuracy scores in the audio domain (specifically Speech Emotion Recognition from Audio files). This raises an interesting question: Will SAL also works in other domains like the image domain?

In order to answer that, we conduct experiments in the image domain (specifically Object Recognition). We make use of the CIFAR10 dataset [15]. The CIFAR10 consists of 60,000 RGB images in 10 classes. As illustrated in Figure 37, we first split the dataset into two subsets. Subset 1 will contain all the instances of the first 5 classes in CIFAR10, and Subset 2 will contain the last 5 classes. We design two Convolutional networks (CNN). The first CNN is used to classify the images in Subset 1, and the second CNN will do the same for images in Subset 2.

![CIFAR10 Dataset](image)

Figure 37 - Splitting of the CIFAR10 Dataset
We are interested in finding out if SAL can help CNN 2. CNN 1 and CNN 2 have the same network architecture. Their architecture is illustrated in Figure 38 above.

We apply SAL on the Dense Layer of the CNNs as depicted in Figure 38. We present the results in Table 39 below.

Table 39 - Results of applying Selective Accelerated Learning on Object Recognition Task

<table>
<thead>
<tr>
<th></th>
<th>CNN 1</th>
<th>CNN 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without SAL</td>
<td>0.5622</td>
<td>0.5618</td>
</tr>
<tr>
<td>With SAL</td>
<td>-</td>
<td>0.5632</td>
</tr>
</tbody>
</table>

In Table 39 above, we observe that the test accuracy of CNN 2 improved after Selective Accelerated Learning. However, the improvement is marginal and does not appear to be statistically significant.

The issue with the experiment is that the Flatten Layer of CNN 1 and the Flatten Layer of CNN 2 cannot be seen as equivalents. This is because the weights learned in the Convolutional Layer of CNN 1 differs from the weights learned in that of CNN 2. Thus, we explore initializing the Convolutional Layer of CNN 2 with the trained Convolutional Layer of CNN 1, and then applying SAL on CNN 2\footnote{I apologise that no further details can be provided on why we set up the experiment as such because it will infringe on the technical disclosure that has been submitted for SAL.}. Please see the footnote on this page.
Table 40 - Results of SAL with CNN 2 being initialised with the trained Convolutional Layer of CNN 1

<table>
<thead>
<tr>
<th></th>
<th>CNN 1</th>
<th>CNN 2 initialised with trained Conv Layer of CNN 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without SAL</td>
<td>0.5622</td>
<td>0.5476</td>
</tr>
<tr>
<td>With SAL</td>
<td>-</td>
<td>0.5476</td>
</tr>
</tbody>
</table>

As shown in Table 40 above, there is no visible improvement in the test accuracy of CNN 2 when we initialize the Convolutional Layer of CNN 2 with the trained Convolutional Layer of CNN 1, and then apply SAL. One possible reason is that CNN 2 is being led to a poor local optimum as a result of its Convolutional Layer being initialised with undesirable weights from CNN 1. This could have led CNN 2 to be insensitive to SAL. This explanation is supported by the fact that the CNN 2’s test accuracy without SAL fell from 0.5618 in Table 39 to 0.5476 in Table 40, which indicates presence of undesirable weights in the Convolutional Layer of CNN 2.

In order to ensure that the Flatten Layer of CNN 1 and the Flatten Layer of CNN 2 can be a pair of fair equivalence, we explore the usage of using pre-trained models to replace the Convolutional Layers in CNN 1 and in CNN 2.

We use the pre-trained MobileNet Network [16] to retrieve a set of features from the CIFAR10 dataset. MobileNet is selected as the pre-trained model to use as it is lightweight compared to the other pre-trained models and suitable for small-scale exploratory experiments. The set of features obtained from MobileNet is then split into Subset 1 and Subset 2 as usual. Subset 1 (1st 5 classes of CIFAR10) is fed into CNN 1 and Subset 2 (last 5 classes of CIFAR10) is fed into CNN 2. The architecture of both CNNs is presented in Figure 39 below. As the MobileNet returns high-dimensional features, we apply Max Pooling to reduce the number of features to 4096.

Figure 39 - Architecture of both CNN 1 and CNN 2 when we use the Pre-trained MobileNet. Note that both CNN 1 and CNN 2 use the same instance of the Pre-trained MobileNet.
Now, the features outputted by the Pre-trained MobileNet for images in Subset 1 should be able to correspond fairly, element-wise, to the features outputted by MobileNet for images in Subset 2. This resolves the issue in the previous experiment whereby the kernel weights of CNN 1 and CNN 2 are different, inhibiting proper functioning of SAL. Next, we apply SAL on the CNNs. We present the results in Table 41 below.

Table 41 - Results of applying SAL on Object Recognition Task with Pre-trained MobileNet features

<table>
<thead>
<tr>
<th></th>
<th>CNN 1 with Pre-trained MobileNet features</th>
<th>CNN 2 with Pre-trained MobileNet features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without SAL</td>
<td>0.8122</td>
<td>0.8122</td>
</tr>
<tr>
<td>With SAL</td>
<td>-</td>
<td>0.8006</td>
</tr>
</tbody>
</table>

Table 41 above shows SAL is unable to improve the test accuracy of CNN 2 when the features used are from the pre-trained MobileNet. This suggests that SAL may not actually work well in the image domain, in particular in the area of the Object Recognition Task. Thus, it is best to constrain the usage of Selective Accelerated Learning to the Audio Domain.
Chapter 5

Evaluation of Techniques against State-of-the-art Performance

In this Chapter, we will use our findings on SAL and CAR in order to build a model capable of outperforming the current state-of-the-art performance for the EmoDB and RAVDESS datasets. Note that due to a pending Technical Disclosure and a potential patent submission, we are unable to describe the SAL and CAR as well as provide much explanations or describe any interesting observations about the results.

5.1 Experimental Set-up

Unlike Section 4.3.2, Section 4.3.2.1, and Section 4.3.2.2, we will be using the entire dataset with a normal 8:2 Training-Test split, and optimizing all hyperparameters (e.g. no longer using just 1 hidden layer) here in order to beat the state-of-the-art performance. We will be building 3 components - Gender Classifier, Male Emotion Classifier and Female Emotion Classifier. For each of the Emotion Classifiers, we used a neural network with 2 hidden layers. The first layer contains 400 neurons, and the second layer contains 250 neurons. Figure 40 below shows the architectures of the 3 individual components. We combined the 3 individual components together to form what we shall refer to as the Stacked Model. We assume that we have the gender information in our training set but not in our test set. Each of the 3 components will take in the PRAAT features of the speech signals as inputs. The batch size used for all 3 components is 64.

Figure 40 - Architectures of the Gender Classifier, Male Emotion Classifier and Female Emotion Classifier

41 The full illustration of the combined architecture cannot be shown due to a pending Technical Disclosure submission.
5.2 Training of the Models

Like in Section 4.3.2, we will first train the Gender Classifier for it to learn to identify the gender of each speech signal. The Gender Classifier will be given the input examples’ PRAAT features as inputs and the gender of the input examples as labels. We train the Gender Classifier for 2000 epochs. After which, the Gender Classifier is able to achieve test accuracy of 98.51%. High gender classification accuracy is essential for the other two components (Male Emotion Classifier and Female Emotion Classifier) to work well.

The Female Emotion Classifier is given only the female training instances (we know which training instances are from the female speaker because we have gender labels for the training set.) However, we assumed that we do not know the gender labels for the testing instances. Thus, the only way to identify which testing instances are female will be to depend on the output of the Gender Classifier. The testing instances which are predicted by the Gender Classifier to be female will be used as the test set for the Female Emotion Classifier. We then train the Female Emotion Classifier. After we are done with training of the Female Emotion Classifier, we do the same for the Male Emotion Classifier. As aforementioned, this overall set-up is also known as the Stacked Model.

5.3 Performance of Stacked Model against the State-of-the-art Performance

The state-of-the-art accuracy for EMODB and RAVDESS are 86.1% and 71.6% respectively based on the paper by Issa et al. in [17]. Simply by splitting the Training and Test instances into the female and male gender models, we achieved results that are not far away from the state-of-the-art performance for EMODB. The results are shown in Table 42a and Table 42b below. Additionally, we used a vanilla fully connected neural network that train on both male and female train instances at the same time, and test on both male and female test instances. The performance of this vanilla neural network is 0.807 and worse than the stacked model and state-of-the-art. This justified our approach to split the train and test sets by gender for EMODB. The same can be said for RAVDESS as the stacked model slightly better than the fully connected neural network.
Table 42a - Results of Stacked Model against the State-of-the-art Performance for EMODB

<table>
<thead>
<tr>
<th></th>
<th>Female instances test acc</th>
<th>Male instances test acc</th>
<th>Overall test acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacked model</td>
<td>0.859</td>
<td>0.82</td>
<td>0.844</td>
</tr>
<tr>
<td>Issa et al. [17]</td>
<td>-</td>
<td>-</td>
<td>0.861</td>
</tr>
<tr>
<td>Fully connected NN</td>
<td>-</td>
<td>-</td>
<td>0.807</td>
</tr>
</tbody>
</table>

Table 42b - Results of Stacked Model against the State-of-the-art Performance for RAVDESS

<table>
<thead>
<tr>
<th></th>
<th>Female instances test acc</th>
<th>Male instances test acc</th>
<th>Overall test acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacked model</td>
<td>0.778</td>
<td>0.667</td>
<td>0.726</td>
</tr>
<tr>
<td>Issa et al. [17]</td>
<td>-</td>
<td>-</td>
<td>0.716</td>
</tr>
<tr>
<td>Fully connected NN</td>
<td>-</td>
<td>-</td>
<td>0.681</td>
</tr>
</tbody>
</table>

5.4 CAR against the State-of-the-art Performance for EMODB

We seek to further improve on our results using CAR.

The effect of applying CAR on Male and Female Emotion Classifiers individually is shown in Table 43a and Table 43b below. We can see that the CAR has led to significant improvement in the test accuracy of both the Male and Female Emotion Classifiers.

Table 43a - Effect of CAR on EMODB

<table>
<thead>
<tr>
<th></th>
<th>Without CAR</th>
<th>With CAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male Emotion Classifier test acc</td>
<td>0.82</td>
<td>0.84</td>
</tr>
<tr>
<td>Female Emotion Classifier test acc</td>
<td>0.859</td>
<td>0.882</td>
</tr>
</tbody>
</table>

Note that the Stacked Model contains both the Male Emotion Classifier and the Female Emotion Classifier.

State of the art for EmoDB

Also State of the art for RAVDESS
Table 43b - Effect of CAR on RAVDESS

<table>
<thead>
<tr>
<th></th>
<th>Without CAR</th>
<th>With CAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male Emotion Classifier test acc</td>
<td>0.667</td>
<td>0.667</td>
</tr>
<tr>
<td>Female Emotion Classifier test acc</td>
<td>0.778</td>
<td>0.791</td>
</tr>
</tbody>
</table>

Table 44a - Results of CAR against the State-of-the-art Performance for EMODDB

<table>
<thead>
<tr>
<th></th>
<th>Female instances test acc</th>
<th>Male instances test acc</th>
<th>Overall test acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacked Model + CAR</td>
<td>0.882</td>
<td>0.84</td>
<td>0.867</td>
</tr>
<tr>
<td>Issa et al. [17] 47</td>
<td>-</td>
<td>-</td>
<td>0.861</td>
</tr>
</tbody>
</table>

Table 44b - Results of CAR against the State-of-the-art Performance for RAVDESS

<table>
<thead>
<tr>
<th></th>
<th>Female instances test acc</th>
<th>Male instances test acc</th>
<th>Overall test acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacked Model + CAR</td>
<td>0.791</td>
<td>0.667</td>
<td>0.733</td>
</tr>
<tr>
<td>Issa et al. [17] 48</td>
<td>-</td>
<td>-</td>
<td>0.716</td>
</tr>
</tbody>
</table>

For EMO-DB in Table 44a above, we observe the overall test accuracy when we apply CAR on both the Male and Female Emotion Classifiers. The resulting overall test accuracy is 0.867, which is significantly higher than the 0.844 (whereby there is no CAR used) recorded in Table 42a. We observe positive but less pronounced improvements for RAVDESS when we applied CAR as the test accuracy improved from 0.726 (no CAR used) in Table 42b to 0.733 (CAR used) in Table 44b.

47 State of the art for EmoDB
48 Also State of the art for RAVDESS
5.5 SAL against the State-of-the-art Performance for EMODB

We also seek to investigate if SAL can help our Stacked model in Section 5.3 improve its test accuracy.

We report the results in Table 45a and Table 45b below. For the Male Emotion Classifier, it appears that SAL does not help with its performance. For the Female Emotion Classifier, we find that SAL is able to improve its test accuracy from 0.859 to 0.8706.

These observations appear to be different from what we observed in Section 4.3.2.2, whereby the Female Model does not gain from SAL. The results in Table 45a and Table 45b instead indicate that it is possible for Female Model to improve by using SAL as well.

There is no need for us to show the overall test accuracy when the Male Emotion Classifier is being applied with SAL since there is no improvement in it. Thus, we only display the overall test accuracy when the Female Emotion Classifier is being applied with SAL in Table 46a and Table 46b below. For EmoDB, the resulting overall test accuracy is 0.852, which is a further improvement from the 0.844 recorded in Table 42a. This supports the idea that SAL is able to help improve the overall performance of our models. We observe similar positive improvements for RAVDESS when we applied CAR as the test accuracy improved from 0.726 (no SAL used) in Table 42b to 0.736 (SAL used) in Table 46b.

Table 45a - Effect of SAL on EMODB

<table>
<thead>
<tr>
<th></th>
<th>Without SAL</th>
<th>With SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male Emotion Classifier test acc</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>Female Emotion Classifier test acc</td>
<td>0.859</td>
<td>0.8706</td>
</tr>
</tbody>
</table>

Table 45b - Effect of SAL on RAVDESS

<table>
<thead>
<tr>
<th></th>
<th>Without SAL</th>
<th>With SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male Emotion Classifier test acc</td>
<td>0.667</td>
<td>0.674</td>
</tr>
<tr>
<td>Female Emotion Classifier test acc</td>
<td>0.778</td>
<td>0.791</td>
</tr>
</tbody>
</table>
Table 46a - Results of SAL against the State-of-the-art Performance for EMODB

<table>
<thead>
<tr>
<th></th>
<th>Female instances test acc</th>
<th>Male instances test acc</th>
<th>Overall test acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male &amp; Female Classifiers with SAL on Female</td>
<td>0.8706</td>
<td>0.82</td>
<td>0.852</td>
</tr>
<tr>
<td>Issa et al. [17] 49</td>
<td>-</td>
<td>-</td>
<td>0.861</td>
</tr>
</tbody>
</table>

Table 46b - Results of SAL against the State-of-the-art Performance for RAVESS

<table>
<thead>
<tr>
<th></th>
<th>Female instances test acc</th>
<th>Male instances test acc</th>
<th>Overall test acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male &amp; Female Classifiers with SAL on Female</td>
<td>0.791</td>
<td>0.674</td>
<td>0.736</td>
</tr>
<tr>
<td>Issa et al. [17] 50</td>
<td>-</td>
<td>-</td>
<td>0.716</td>
</tr>
</tbody>
</table>

5.6 Combining CAR and SAL against the State-of-the-art

Lastly, we combined CAR with SAL to see if we could further improve our results and increase the gap between our results and that of the State-of-the-art. We apply SAL on the Female Emotion Classifier and use CAR on both Female and Male Emotion Classifiers. We report the results in Table 47a and Table 47b below.

Table 47a - Results of CAR and SAL against the State-of-the-art Performance for EMODB

<table>
<thead>
<tr>
<th></th>
<th>Female Classifier, CAR + SAL</th>
<th>Male Classifier, CAR</th>
<th>Overall test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacked Model31</td>
<td>0.894</td>
<td>0.84</td>
<td>0.874</td>
</tr>
<tr>
<td>State-of-the-art</td>
<td>-</td>
<td>-</td>
<td>0.861</td>
</tr>
</tbody>
</table>

49 State of the art for EmoDB
50 Also State of the art for RAVDESS
51 Gender Classifier stacked on top of Emotion Classifier
Table 47b - Results of CAR and SAL against the State-of-the-art Performance for RAVDESS

<table>
<thead>
<tr>
<th></th>
<th>Female Classifier, CAR</th>
<th>Male Classifier, CAR + SAL</th>
<th>Overall test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacked Model(^{52})</td>
<td>0.791</td>
<td>0.674</td>
<td>0.736</td>
</tr>
<tr>
<td>State-of-the-art</td>
<td>-</td>
<td>-</td>
<td>0.716</td>
</tr>
</tbody>
</table>

From Table 47a above, we observe that combining CAR with SAL have led to a further improvement in the performance of the stacked model. We achieve a test accuracy of 0.874 in EMODDB, which is significantly higher than the state-of-the-art performance of 0.861. A similar improvement can be seen with RAVDESS dataset in Table 47b whereby we achieve a test accuracy of 0.736 against the state-of-the-art performance of 0.716.

The results thus suggest that using CAR, or SAL, or a combination of both, can lead to further improvement in test accuracy.

### 5.7 Comparison of performances of Stacked Model with CAR and SAL with the existing approaches

Current state-of-art SER performance on EMO-DB and RAVDESS was reported by Issa et al [17]. Issa extracted the PRAAT features and employed an one-dimensional Convolutional Neural Network for speech emotion recognition. Their results are presented in Table 48. Other recent methods include Wen’s [18] ensemble of deep belief networks and support vector machines to recognise emotions. Zamil’s [19] applied a Logistic Model Tree (LMT) onto Mel Frequency Cepstrum Coefficient (MFCC) features extracted from speeches to detect emotions within the speech. Lastly, Jalal [20] made use of a bidirectional long short-term memory network (BLSTM), CNN and Capsule networks to carry out speech emotion recognition in RAVDESS.

Table 48 - Comparison of accuracies (%) of speech emotion recognition by Stacked Model with CAR and SAL with existing state-of-art techniques.

<table>
<thead>
<tr>
<th>Method</th>
<th>Emo-DB</th>
<th>RAVDESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacked Model with CAR and SAL</td>
<td>87.4</td>
<td>73.6</td>
</tr>
<tr>
<td>Issa et al. [17]</td>
<td>86.1</td>
<td>71.6</td>
</tr>
<tr>
<td>Wen et al. [18]</td>
<td>82.3</td>
<td>-</td>
</tr>
<tr>
<td>Zamil et al. [19]</td>
<td>64.5</td>
<td>70.0</td>
</tr>
<tr>
<td>Jalal et al. [20]</td>
<td>-</td>
<td>69.4</td>
</tr>
</tbody>
</table>

\(^{52}\) Gender Classifier stacked on top of Emotion Classifier
Chapter 6

Conclusions and Future Work

6.1 Conclusions

For the first part of this project, we showed that using the PRAAT features of the audio signals is generally a better way to obtain gender-invariant representations as compared to using just MFCC or Audio chunks. With PRAAT features, we observe that a combination of supervised loss, Independence Loss and Triplet Loss can help in generating better gender-invariant representations of the input signals. We also demonstrated the potential of using DEC and Soft Nearest Neighbour Loss to do the same.

In the second part of the project, we tried using MTL techniques with Emotion Classifier and Gender Classifier as a means to obtain better emotion classification scores. The different techniques such as Cross-stitched networks, Correlated information removal MTL etc. produced positive results. But the results are generally too insignificant to warrant further extension or exploration. Through the MTL experiments, however, we discovered the usefulness of separating the male and female instances into different emotion classifiers. We further explore this in the last part of this project.

In the last part of the project, we built a stacked model consisting of a gender classifier at the front, and two emotion classifiers (male and female) at the back. The results show that separating the emotion classification into male emotion classification and female emotion classification actually led to an improvement in the overall emotion classification. This may imply that the PRAAT features of the male and female instances are rather distinct from each other. Thus if we simply use a single emotion classifier to train on both male and female instances, the difference between male and female instances may confuse the emotion classifier during training.

We then introduced two different techniques - 1. Class-based and Re-centred (CAR) Data Augmentation (CAR) and 2. Selective Accelerated Learning (SAL). We ran experiments to prove their usefulness. Interestingly, the two techniques seem to help in emotion classification even when the speeches are in different languages as demonstrated in Section 4.4. However, a drawback is that our experiments seem to indicate that SAL will not be able to work well in the Image domain with image features. Thus, it is better for us to focus on SAL in the Audio domain. It is not known, however, about the usefulness of CAR in the Image Domain and this remains a potential research area.

Finally, we proved that our stacked model, together with CAR and SAL, are able to outperform the current state-of-the-art performances for the EMODB and RAVDESS datasets by a significant margin.
6.2 Recommendation in Future Work

Future work could explore whether a combination of Triplet Loss and SNNL would further improve the gender-invariant representations of the input signals. There are also other losses that could be explored such as contrastive loss and lifted struct loss.

Another potential area of research is whether CAR could be used in other domains such as the Image domain. Also, it would also be interesting to investigate whether SAL and CAR would work if the audio features are not PRAAT. For example, whether SAL and CAR would work on audio chunks containing MFCC coefficients.

The stacked model has worked well in the EMODB and with PRAAT features. But it would be interesting to see if the stacked model would work in other contexts such as in Cifar100 etc.
Reflection on Learning Outcome Attainment

Prior to this FYP, I had taken the Neural Network and Machine Learning modules. Despite that, it seems that I was not fully prepared to undertake independent research on neural networks due to the perceived lack of experience in this field.

This project really helped me to gain experience and understanding of the neural network, both in terms of theory and code implementation. I feel at ease with generating novel ideas to resolve or improve a neural network model. At the same time, I am confident of building and modifying neural networks from scratch using Tensorflow.

The idea of exploring the hidden representations in neural networks, and how we could exploit or improve these representations is captivating and may prove to be important in future development of this field. Different researchers seem to have a different opinion on what the hidden representations really mean. It is, of course, an open question as of now.

But one view is that the hidden representations at the shallow layers corresponds to the lower level features of the input instances, and the deeper layers corresponds to the higher level feature of input instances. For example, in a CNN model, the lower level layers would be edges or edge images, and the higher level layers would be a more human-like or higher-level interpretation of the input instances.

Another view is that the hidden representations simply represent random numbers that lead to the model achieving its goal, whatever that goal may be. Both are equally plausible views, and I doubt that they are actually mutually exclusive.

This project mainly revolves around representations and it leads me to ponder about how crucial representations really are in a neural network. I would love to continue exploring on the topic of representation learning, its meanings, and applications.
References


Appendix

There are quite a few ideas which we would like to explore in the future. Many of these ideas are probably explored in a somewhat shallow manner during the term of this project.

The first idea is the usage of GAN, and specifically Cyclic GAN. We would like to explore whether by converting male and female instances into a gender-neutral representation using Cyclic GAN would help in Speech Emotion Recognition.

We did conduct experiments with Cyclic GAN using the above idea. Unfortunately, there is a lack of positive results. It could be due to two reasons: 1. the Cyclic GAN implemented is too rudimentary and too basic to work. 2. the idea does not work at all. Either way, we did not continue exploring in this direction due to the lack of time, and that this idea seems to diverge too far away from the topic of this project.

Another idea that we could explore is the aforementioned paper written by Frosst, N., Papernot, N. and Hinton, G. in [12]. The paper argues that instead of trying to maximising the distance between instances of different classes as done by Triplet Loss and Soft Nearest Neighbor Loss, we should actually minimizing the distance in the front/first few layers of the neural network.

The intuition behind this is that the front layers of the neural network should be looking for the common patterns and features of the input instances. Then only at the last few layers, then the neural network should make the distinction between the input instances based on their classes.

The paper demonstrated that Triplet Loss is not suitable for this idea, and that Soft Nearest Neighbor Loss should be used instead. Thus, if we have additional time for this project, we could have explored minimizing the Soft Nearest Neighbor Loss in the front/first few layers of the neural network and maximizing the Soft Nearest Neighbor Loss in the last few layers.