NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SCSE20-0439

SPEAKER-INVARIANT SPEECH EMOTION RECOGNITION WITH DOMAIN ADVERSARIAL TRAINING

Submitted in Partial Fulfillment of the Requirements
for the Double Degree Programme of Bachelor of Business and
Bachelor of Engineering (Computer Science)

by

Bryan Leow Xuan Zhen

Supervisor: Professor Jagath C Rajapakse

School of Computer Science and Engineering
2021

Abstract

Recent advances in technology have given birth to intelligent speech assistants such as Siri and Alexa. While these intelligent speech assistants can perform a myriad of tasks just from the end users' voice command, they lack the capability to recognize human emotions when formulating a response — a feature that would promote more ingenious usage for such speech assistants. Since such a Speech Emotion Recognition (SER) system would be used by the general population, it is necessary to derive speaker-invariant representation for the SER system.

In this project, we use Domain Adversarial Training (DAT) in deep neural network to learn representation that is invariant to speaker characteristics. DAT was used for domain adaptation, in which data at training and test time come from similar but different distributions or speakers. Recognising that speaker invariant SER can be framed as a domain adaptation problem, we explore the use of DAT in this project to derive speaker-invariant representations for SER and observe if they perform better than the representations formed without DAT.

DAT network for speaker-invariant emotion recognition (SIER) tasks consist of an encoder, an emotion classifier, and a speaker classifier. By having a Gradient Reversal Layer (GRL) between the encoder and the speaker classifier, the emotion representation learned will be independent of speakers. DAT encoder in existing literature has typically been limited to 1D Convolutional Neural Network (CNN) with Recurrent Neural Network (RNN) architectures. In contrast to such architectures which use 1D filters to learn features along a single dimension, this paper investigates DAT encoders of 2D CNN with RNN architecture which use 2D filters to learn features along two dimensions. We also investigate Log Mel Spectrograms (LMS) and Mel Frequency Cepstral Coefficients (MFCC) features for 2D CNN with RNN DAT encoders. Our experimental results on Emo-DB and RAVDESS datasets show that MFCC features with 2D CNN with RNN DAT encoders performs better than features and encoders that relies on 1D filters.

Acknowledgement

First and foremost, I would like to express my deep and sincere gratitude to my research supervisor, Professor Jagath C Rajapakse for giving me this opportunity to do research. His profound belief in my abilities and continual encouragement has been instrumental for me to complete this research successfully. I would also like to extend my sincere thanks to Dr. Liyanaarachchi Lekamalage Chamara Kasun and Mr. Ahn Chung Soo for their mentorship. Their extensive knowledge in the field has played a critical role in how I shaped my experimental methods and critiqued my results. Special thanks also go to SCALE@NTU for allowing me to conduct my experiments on their server. It is with the help of these personnel that allowed me to submit my first ever paper to INTERSPEECH 2021 while doing this FYP. Finally, I would like to acknowledge with gratitude, the support and love of my parents and brothers who kept me going when the going gets tough.

Table of Contents

List of figures

List of Code Snippets

1. Introduction

1.1 Background

Speech Emotion Recognition (SER) aims at classification of speech signals into the class or label of emotions expressed. SER algorithms can be categorised as fully connected neural networks [1]–[4] which use high level features such as The Interspeech para-linguistics challenge 2010 features[5] and deep learning approaches which use low level features such as Log Mel Spectrograms (LMS) and Mel Frequency Cepstral Coefficients (MFCC). The advantage of the latter approach over the former is that utilising deep learning architectures fed with low level features do not require the time-consuming process of creating high level features. As such, numerous research in SER with low level features have been conducted with Convolutional Neural Network(CNN) and Recurrent Neural Network [6]–[10] for emotion prediction task.

In SER, experiments are generally classified as Speaker Dependent Emotion Recognition (SDER) and Speaker Independent or Speaker Invariant Emotion Recognition (SIER). SIER tasks are generally more difficult since the training and testing data comes from different speakers [11]. Hence, addressing the problem of SIER tasks is one area of active research. Since the aim of SIER is to create a model from speech data spoken from a set of speakers and apply the model to predict emotions from speech from a different set of speakers, SIER algorithms must learn a speaker invariant representation, where speakers' identity is removed.

SIER tasks typically have limited number of training data due to the difficulty of collecting and labelling speech emotion data. It also faces the issue of variability since the environment the training data is collected is different from the environment the predictions are done. To address the issue of limited number of data and variability, adversarial learning approaches have been proposed.

Adversarial learning based on Generative Adversarial Network (GAN)[12] is introduced in SIER tasks to generate training data [13]. Traditional GAN architectures use a random Gaussian vector as a seed to generate data. Hence these GAN architectures model the data generation process as a Gaussian distribution. However, due to the high variability of speech emotion data, it is not possible to model speech emotion data as a simple Gaussian distribution[14]. Hence speech emotion GAN approaches seed a linearly mixed speech emotion data to generate data. Another approach of generating speech emotion data is to use an autoencoder to generate latent vector[15] and use this latent vector added with random Gaussian noise as a seed to a GAN to generate speech emotion data. This approach has been shown of capable of generating speech emotion belonging to a specific category such as anger or happy. Models trained with the generated and original data has shown to perform better than models trained with only original data.

Adversarial learning based on Domain Adversarial Training (DAT) has also been used to resolve the disparity between training and prediction environment. Experiments have shown that DAT network can remove the variability in the environment and predicts well on unseen target domain data even though the model is trained on source domain data. For example, given two SER datasets (one with German speakers, another with English speakers), a typical machine learning algorithms that is trained on German speech emotion data will have difficulty in predicting emotion of English speech emotion data. DAT can then be applied to remove the variability caused due to such domain differences [16]. This is achieved by using an encoder, emotion classifier and domain classifier with gradient reversal. Domain classifier aims to remove the variability of the domain. This same concept of learning a domain invariant representation can be extended to learning a speaker invariant representation by changing the domain classifier to speaker classifier with gradient reversal [17], [18].

As mentioned earlier, low level features such as LMS and MFCC have been conducted with deep learning architectures before. The popularity of LMS and MFCC in SER deep learning approach can be attributed to their usage of the logarithmic Mel-scale where equal distance in pitch sound equally distant to the listener. This is important because humans do not perceived frequency on a linear scale and can distinguish low frequencies at a better resolution than high frequencies, i.e. a human can tell apart a 500Hz and 1000Hz signal but cannot tell the difference between a 1Khz and 1.5kHz signal. Hence, LMS and MFCC features aim to capture this non-linear perception of sound with the frequencies are spaced according to perception. LMS and MFCC represents speech as a set of frequencies that vary along with time. The number of frequencies captured can be determined by the user. The main difference between LMS and MFCC features is that LMS represents the magnitude of a frequency, while MFCC represents the energy of a frequency.

LMS and MFCC captures the frequency information along the time axis. However, existing DAT encoders use a 1D CNN with RNN architecture where the 1D CNN learn short term temporal features and the RNN learn long term temporal features from the short-term temporal features learned by the 1D CNN. However, such short-term temporal features capture only one frequency component and will not be suitable for SIER tasks as speakers voice is composed of multiple frequency components. In another word, 1D CNN with RNN architectures is not suitable for capturing rich frequency information in speech since they capture only temporal information. For capturing the rich frequency information in speech, the architecture must be able to capture both temporal and frequency information.

1.1 Project Objective and Scope

This project aims to create a model architecture for Speaker Independent Emotion Recognition (SIER) task by using Domain Adversarial Training (DAT) as a framework. The model architecture should yield better performance than the existing DAT encoder (1D CNN with RNN encoder) doing SIER. The effect of different rate of domain adaptation during DAT on the performance of the model architecture in SIER tasks and the usefulness of LMS and MFCC for SIER tasks with DAT will also be compared and investigated. The performance of the model architecture created will also be compared to existing methods using DAT.

1.2 Summary of Work Done

We conduct our experiments on 2 different datasets:

1. Berlin Database of Emotional Speech(Emo-DB) [19]
2. Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) [20]

The name of each .wav file contains information on the speaker identity and emotion. We first extract these information as they are necessary for training. We also do pre-process of these audio waveforms and convert them to their respective LMS and MFCC features. The features to be fed into encoders.

We design 2D l-CNN with Bidirectional Gated Recurrent Unit (Bi-GRU) structures, where l represents the number of CNN in the encoder and Bi-GRU is a type of RNN. The rationale for selecting such an encoder structure is because in addition to temporal features captured with 1D CNN, the 2D CNN will also capture multiple frequency useful for SIER task before passing onto the RNN.

Each DAT network is composed of an encoder, a speaker classifier, and an emotion classifier. We ran different experiments with different encoders but use the same speaker classifier and emotion classifier. For each encoder, multiple experiments are also tried out with different domain adaptation parameters (DAP). The DAP is basically a scheduler that controls the rate and degree of domain adaptation during DAT.

We then compared our 2D l-CNN Bi-GRU DAT encoders to the existing DAT encoders doing SIER with cross-validation and testing. To ensure fairness in comparisons, we implemented those encoders and perform cross-validation and testing on them like how we would have performed on our 2D l-CNN Bi-GRU DAT encoders.

A paper was also written to be submitted to INTERSPEECH 2021.

1.3 Summary of Results

Our experiments show that our 2D l-CNN Bi-GRU DAT encoders perform significantly better than the existing DAT encoders used in SIER. Interesting to note is that the existing DAT encoders use 1D CNN in part of their encoders. This confirms our beliefs that the failure to capture the rich frequency have impedes the ability of such encoders to do well in SIER tasks.

We also found out that different rate and degree of domain adaptation during DAT affects the performance of our encoder, hence it is crucial to select a good Domain Adaptation Parameter(DAP) value. Unfortunately, it is not that easy a task. Too high a DAP would result in too much information being removed, but too low a DAP would end up with the model still depending on speaker-related deep features. Either case could result in the same encoder having a performance that is worse than without DAT.

Our experiments also show that MFCC performs better than LMS in SIER tasks.

1.4 Organization of the Report

Section 1 introduces the project. It contains some background information on SER , such as the features use in SER, the challenges of SIER tasks and how adversarial learning has been used in SER. It also contains the project objective and scope and gives a summary of work done and result.

Section 2 provides a background to the deep learning architectures and deep learning techniques used in this project. It will also discuss DAT and the audio features such as LMS and MFCC as they formed the backbone of this project.

Section 3 goes through the methodology we employed in this project in detail.

Section 4 shows all our experiments and discussion of the results.

Section 5 shows an application we have developed for demonstration purposes.

Section 6 covers the conclusion of our project.

2. Background Concept Review

2.1 Deep Learning Architectures

2.1.1 Artificial Neural Network (ANN)

An Artificial Neural Network (ANN) is a computation model consisting of layers of nodes known as artificial neurons. Its architecture is inspired from the structure of the human brain. The most basic type of ANN is known as the Feed Forward Network (FFN).

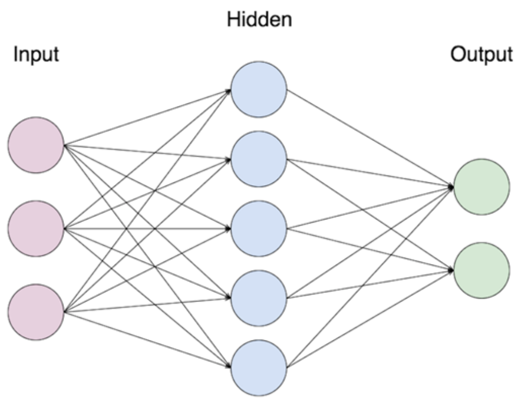


Figure 1 A 3-layer feedforward network

Figure 1 shows a 3-layer FFN. A FFN generally has 3 types of layers: input layer, hidden layer(s) and output layer. The input layer is made up of a collection of input nodes. These input nodes do not perform any computation and is merely responsible for passing the input into the hidden nodes. The number of input nodes in the input layer depends on the number of dimension of the input data. The hidden nodes collectively form the hidden layer(s). Here, computations are performed, and information are transferred from the input layer all the way to the output layer in a process known as forward propagation. A FFN can have zero or more hidden layers. In the case where it has no hidden layer, the FFN is known as a single layer perceptron. In the case where it has one or more hidden layer, it is known as Multi-Layer Perceptron. The output nodes collectively form the output layers. For classification problem, the number of output nodes depends on the number of classes to be predicted by the FFN.

In Figure 1, we can see that there are lines connecting the nodes between the layers. These are the weights and bias in the FNN, and they are randomly initialised. These parameters between any 2 layers (X,Y) can be represented as matrix of size (*x,y*) where *x* represents the number of nodes in layer X and *y* represents the number of nodes in layer Y. During forward propagation, the *i*-dimensional inputs are fed into *i* input nodes, hence it can be treated as a vector of size *i*. Given a hidden layer of *h* nodes, this input vector will undergo matrix multiplication with a matrix of size (*i,h*) to give a hidden vector of size *h*. All the hidden nodes will then be activated with an activation function before undergoing another matrix multiplication with a matrix of size (*h,o*) to give a output vector of size *o*. Depending on the task at hand, the output nodes might or might not be activated with an activation function. The output will be compared with the dataset labels.

To train an ANN means to minimise the loss function which takes in the predicted outputs of the ANN and the dataset labels. This loss function is typically a multi-dimension function that depends on weights and bias parameters. During training process, the gradients of the loss function with respect to the parameters from the output layer of the ANN is first computed and adjusted. Through chain rule, the gradients of the different parameters of the ANN are then computed and adjusted recursively from the last few layers all the way to input layer. This process is known as backpropagation and allow the weights and bias parameters to be gradually adjusted through gradient descent with the aim of finding an optimal set of weight and bias parameters that minimise the loss function. The equation of gradient descent is given by:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \nabla_{\boldsymbol{W}} J$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \nabla_{\boldsymbol{b}} J$$

where $\boldsymbol{W}$ represents the weights, $\boldsymbol{b}$ represents the bias, $\nabla_{\boldsymbol{W}} J$ represents the gradient of the loss function with respect to $\boldsymbol{W}$, $\nabla_{\boldsymbol{b}} J$ represents the gradient of the loss function with respect to $\boldsymbol{b}$ and α represents the learning rate which is a scalar hyperparameter use to control the rate of updating the $\boldsymbol{W}$ and $\boldsymbol{b}$ parameters.

The activation function and loss function will be discussed in a later section.

2.1.2 Convolutional Neural Network (CNN)

The Convolutional Neural Network (CNN) is a type of ANN. CNN is most commonly used in image recognition. Compared to FFN where a neuron at one layer is connected to all the neurons in the next layer, CNN restrict the number of connections from one layer to the next layer, achieving local connectivity between neurons of adjacent layers. Due to the local connectivity, the receptive fields of the neurons are limited, with the receptive field a neuron at the later layers making up from the receptive fields of neurons connecting to it at the earlier layer. The sparse local connectivity saves computational time for signals of larger resolutions. In CNN, each filter is replicated across the entire visual field. These filters share the same weight and bias and form a feature map, which represent the activations of the neurons at the later layers, corresponding to individual filters.

As depicted in Figure 2, a CNN typically has 3 layers: Convolutional Layer, Pooling Layer and Fully Connected Layers.

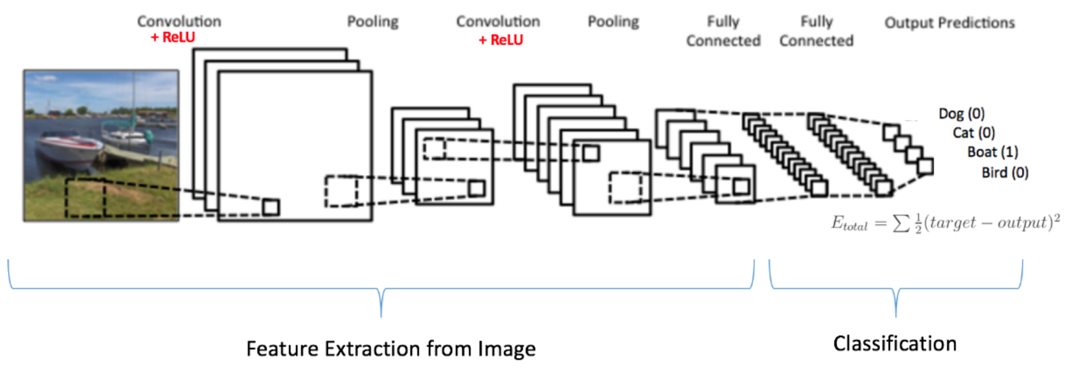


Figure 2 Typical structure of a CNN

Convolutional operation is performed in the convolutional layer. At the start of the training, the weights and bias of the filters are first randomly initialised. By convolving a filter with the input and performing dot products on them, we can get the respective convolved feature or feature map from that filter. An activation function can also be used during the convolution. Note that one filter will produce one feature map. Figure 3 shows an example of the output from the convolution operation. For simplicity, we assume a window size of 2 x 2 for the filter, stride (the distance in which the output is subsampled) of 1, bias of 0 and used a linear activation function. Just like any ANN, the filters' weights will be updated during backward propagation.

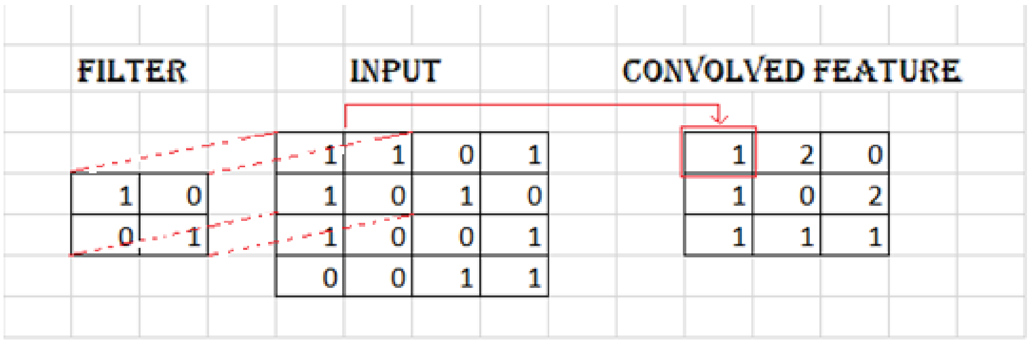


Figure 3 Convolution operation in CNN

The pooling layer is used to reduce the dimensions of the convolution layer. There are 2 types of pooling operation: Max pooling or average pooling. In the pooling area, each convolved feature is pooled by taking either the maximum or average. Figure 4 shows an example of the output from the pooling operation. We assume a window size of 2 x 2 for the pooling operation, stride of 1. Note that activation function is not relevant for pooling operation.

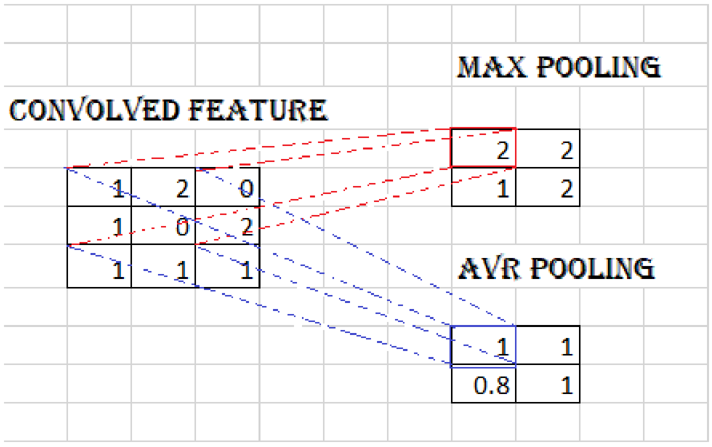


Figure 4 Pooling Operation in CNN

As we can observed, during convolutional operation and pooling operation, the resulting feature map becomes smaller. The deeper the CNN, the smaller the feature map will become. This is not ideal for a deep neural network. To address this concern, we can either use smaller window size or perform padding on the boundary.

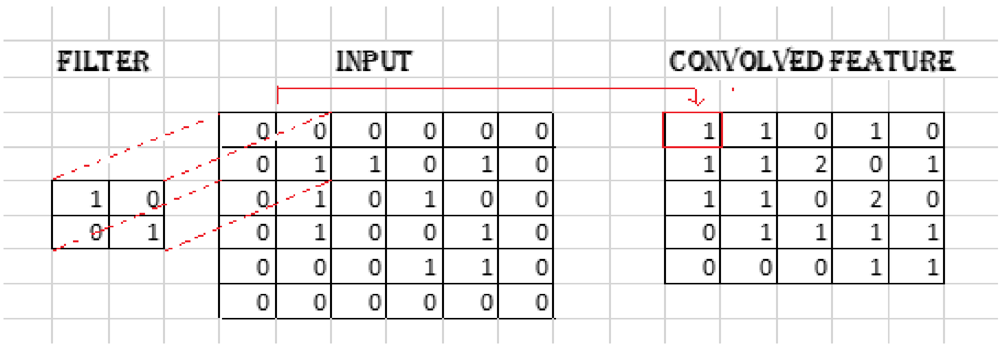


Figure 5 Padding on the Boundary for CNN

As illustrated in Figure 5, padding on the boundary of the input can avoid shrinking the spatial extent of the network rapidly. This can present the representation from shrinking with depth and allow us to make deeper CNN. While Figure 5 only illustrates padding on convolutional operations, it can only be done with pooling operation.

The pooling layer is usually followed by another convolutional layer or a fully connected network. Since the fully connected network is just FFN as mentioned earlier, it would not be elaborated.

2.1.3 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a variant of ANN that is designed to process sequential information. It has memory that allows it to capture information about what has been processed so far. For this reason, RNN is commonly used when it comes to sequential information like time series, financial data, speech and text. Compared to FFN, RNN utilises feedback connection that enables it to learn temporal information of sequences. This idea is illustrated clearly in Figure 6.

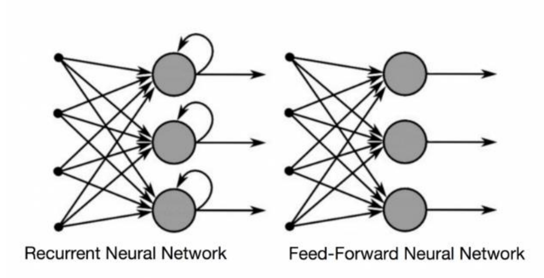


Figure 6 Comparison between RNN and FNN

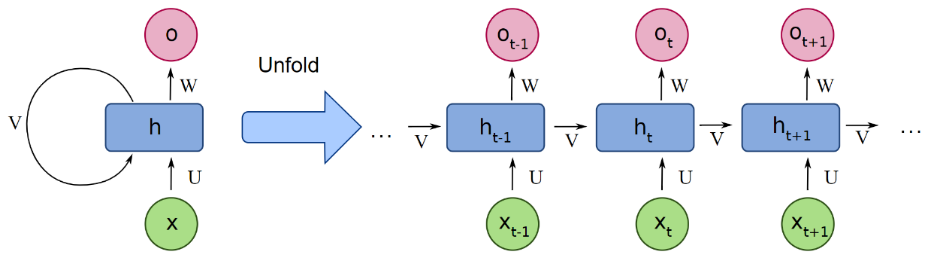


Figure 7 Unfolded structure of a vanilla RNN

Figure 7 shows an example of a vanilla RNN, where:

$\boldsymbol{x}$: *input of sequential data*
$\boldsymbol{h}$: *hidden state*
$\boldsymbol{O}$: *sequence output*
$\boldsymbol{U}$: *weight vector that transforms the raw inputs to the hidden layer*
$\boldsymbol{V}$: *recurrent weight vector connecting previous hidden layer output to hidden input*
$\boldsymbol{W}$: *weight vector of the output layer*
$\boldsymbol{b}$: *bias connected to hidden layer*
$\boldsymbol{c}$: *bias connected to the output layer*

By considering the unfolded structure, we observe that the t^{th} hidden state $\boldsymbol{h_t}$ is updated by the t^{th} sequence $\boldsymbol{x_t}$ of an input sequence and the previous hidden state $\boldsymbol{h_{t-1}}$. It then outputs the t^{th} sequence for the output $\boldsymbol{o_t}$. We can represent a one hidden layer RNN with the following equations:

$$\boldsymbol{h}_t = \phi(\boldsymbol{U}^T \boldsymbol{x}_t + \boldsymbol{V}^T\ \boldsymbol{h}_{t-1} + \boldsymbol{b})$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^T \boldsymbol{h}_t + \mathbf{c})$$

where ϕ represents the tanh hidden-layer activation function and σ represents the linear or softmax output-layer activation function. Figure 8 shows the basic RNN cell.

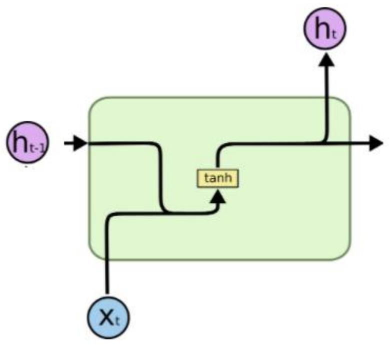


Figure 8 Basic RNN Cell

In the case of RNN, the backpropagation algorithm used for training is given a special name known as Back Propagation Through Time (BPTT). It works the same way as backpropagation in FFN by applying the chain rule on the RNN and updating the weights going from the last to first layer. The only difference is that now, the gradients propagate backward from 2 directions: top-down direction and reverse sequence direction on the unfolded structure of RNN.

While RNN has demonstrate its ability to solve sequential problem, its performance to capture long term dynamics and dependencies is far from satisfactory. This is because during BPTT, the gradient can end up multiplied by the weight matrix associated with the connections between the neurons of RNN multiple time, leading to vanishing gradient problem where the learning become very small in the case of if the weights in the matrix are small or exploding gradient problem where the learning will diverge if the weights in the matrix are large. To tackle this problem, Gated RNNs such as Long Short-Term Memory (LSTM) Unit and Gated Recurrent Unit (GRU) are proposed.

2.1.4 Long Short-Term Memory (LSTM)

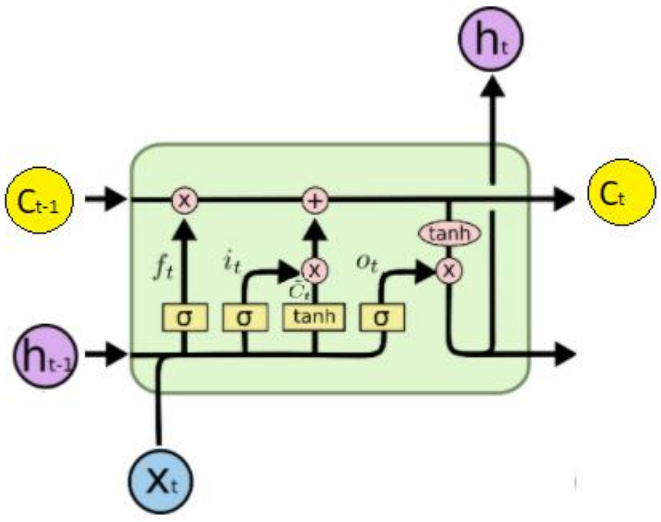


Figure 9 Long Short-Term Memory (LSTM) Unit

Instead of having only one single neural network layer (tanh) as in the case of a basic RNN unit, a LSTM unit has additional 3 hidden sigmoid neural network layers as shown in Figure 9.The sigmoid hidden neural net layers and the pointwise multiplication operations form what we called the gates. Hence, there are 3 different type of gates in a LSTM unit. The interaction of the 4 hidden layers in the LSTM allows it to learn when to forget and update the hidden states when fed with newer information.

Most important to the LSTM is the cell state c_t. While the hidden state h_t is designed to capture the short-term dependencies, the cell state c_t is designed for the purpose of capturing the long-term dependencies.

As mentioned earlier, there are 3 different type of gates in a LSTM unit. They are the forget gate, input gate and the output gate. Through the sigmoid layer in the gates, the LSTM can control how much each component of the information can be added to or removed from the cell state c_t. The sigmoid layer can output value ranging from 0 to 1, describing how much of each component should be added to or removed from the cell state c_t.

The forget gate makes up the first branch of the LSTM unit. It can be written as:

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{U}_{\boldsymbol{f}}^{\boldsymbol{T}}\boldsymbol{x}_t + \boldsymbol{V}_{\boldsymbol{f}}^{\boldsymbol{T}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f\right)$$

where $\boldsymbol{f}_t$ controls how much of c_{t-1} is remember.

The input gate controls what new information can be stored in the cell state. It is made up from the 2nd and 3rd branches of the LSTM unit. The sigmoid input gate layer decides on which value to be updated while the tanh layer creates a vector of new candidate values $\widetilde{\boldsymbol{c}_t}$ that could be added to the state. They can be written as:

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{U}_{\boldsymbol{i}}^{\boldsymbol{T}}\boldsymbol{x}_t + \boldsymbol{V}_{\boldsymbol{i}}^{\boldsymbol{T}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i\right)$$
$$\widetilde{\boldsymbol{c}_t} = \phi(\boldsymbol{U}_{\boldsymbol{c}}^{\boldsymbol{T}}\boldsymbol{x}_t + \boldsymbol{V}_{\boldsymbol{c}}^{\boldsymbol{T}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c)$$

The cell state is then calculated as followed:

$$\boldsymbol{c}_t = \widetilde{\boldsymbol{c}_t} \odot \boldsymbol{i}_t + \boldsymbol{c}_{t-1} \odot \boldsymbol{f}_t$$

The rest of the branches make up the output gate. The output gate determines whether the state of the memory cell will influence other neurons and can be written as:

$$\boldsymbol{o}_t = \sigma(\boldsymbol{U}_{\boldsymbol{o}}^{\boldsymbol{T}}\boldsymbol{x}_t + \boldsymbol{V}_{\boldsymbol{o}}^{\boldsymbol{T}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o)$$

The hidden state $\boldsymbol{h}_t$ is then updated as followed:

$$\boldsymbol{h}_t = \phi(\boldsymbol{c}_t) \odot \boldsymbol{o}_t$$

2.1.5 Gated Recurrent Units (GRU)

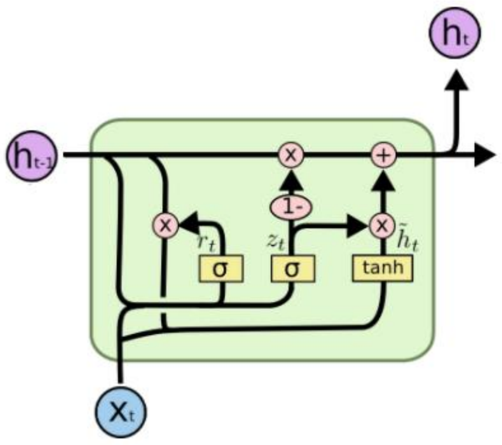


Figure 10 Gated Recurrent Unit(GRU)

Figure 10 shows a Gated Recurrent Unit (GRU). Just like the LSTM, GRU is used to tackle the problem of exploding or vanishing gradient problem that that a vanilla RNN faced. The main difference between LSTM and GRU is that the GRU does not maintain a cell state and that it only has 2 gates as compared to 3 gates in LSTM.

The 2 gates in GRU are the reset gate and the update gate. They work simultaneously to control how much information to be forgotten or added.

The reset gate can be written as:

$$\boldsymbol{r}_t = \sigma(\boldsymbol{U}_r^T \boldsymbol{x}_t + \boldsymbol{V}_r^T \boldsymbol{h}_{t-1} + \boldsymbol{b}_r)$$

It controls how much of the information from the previous hidden state $\boldsymbol{h}_{t-1}$ flows through.

The update gate can be written as:

$$\boldsymbol{z}_t = \sigma(\boldsymbol{U}_z^T \boldsymbol{x}_t + \boldsymbol{V}_z^T \boldsymbol{h}_{t-1} + \boldsymbol{b}_z)$$

It controls how much new information should be added to the previous state to generate the new hidden state.

The candidate hidden activation function can then be written as:

$$\widetilde{\boldsymbol{h}_t} = \phi\big(\boldsymbol{U}_h^T \boldsymbol{x}_t + \boldsymbol{V}_h^T (\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h\big)$$

Finally, the hidden state $\boldsymbol{h}_t$ can be updated as followed:

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \widetilde{\boldsymbol{h}_t}$$

with the 1st term $(1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1}$ used to forget things from the past adaptively and the 2nd term $\boldsymbol{z}_t \odot \widetilde{\boldsymbol{h}_t}$ used to adaptively add new information.

The advantages of LSTM and GRU are numerous. Apart from tackling the exploding gradient and vanishing gradient problem, compared to the vanilla RNN which overwrites it content at each time step, the gating mechanism in GRU and LSTM allows the unit to decide whether to keep the existing memory at each time step. This allows them to detect more important features from an input sequence and carries the features over a longer distance, capturing potential long-distance dependencies. Due to this, they are more commonly used than vanilla RNN.

2.2 Deep Learning related functions

2.2.1 Activation functions

Activation functions are essential tools in deep learning. An activation function will decide whether a neuron should be activated or not by calculating the weighted sum and biases with it. Backpropagation of errors is only possible with the help of activation function as the gradients are supplied along with the error to update the weight and biases. Non-linear activation functions introduce non-linearities into the ANNs which allow them to solve more complex problems. There are many different types of non-linear activation functions, but we will only be mentioning those used in this project.*2.2.1.1 Rectified Linear Unit (ReLU)*

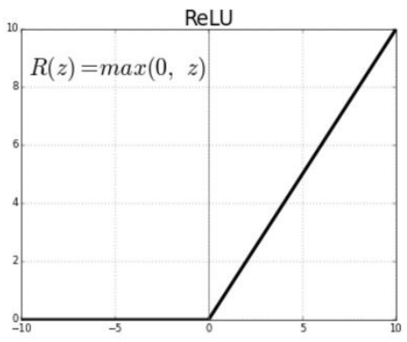


Figure 11 Rectified Linear Unit (ReLU) activation function

The first activation function we would be discussing is the Rectified Linear Unit (ReLU) activation function. As illustrated in Figure 11, It is a piecewise activation function that will output the input directly if positive or zero if negative. Compared to more traditional activation function such as sigmoid and tanh, it does not suffer from vanishing gradient problem and can output true zero values when the inputs are negative. This enables it to create sparse representation that can accelerate learning and simplify representational learning models. Its linear behaviour when the input is positive also makes it easier to optimise. Due to its simplicity and efficiency, it has become the default activation function for deep learning.

Despite the many benefits that ReLU offers, it does have its limitations. Since all negative values will become 0, it leads to a problem known as "dying ReLU". A ReLU neuron is dead if it is stuck in the negative side and always output 0. As the derivative of ReLU in the negative range is also 0, once it gets negative, it is unlikely to recover. A dead neuron is thus essentially useless and will no longer have any role in training the model. To solve this problem, we can consider variants of ReLU like the Exponential Linear Unit (ELU).

2.2.1.2 Exponential Linear Unit (ELU)

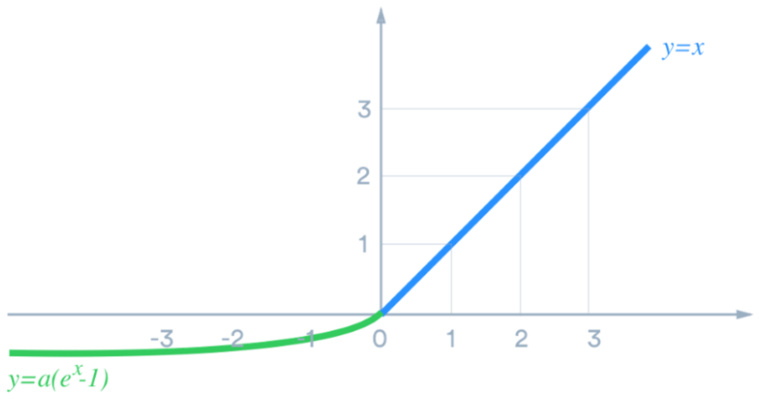


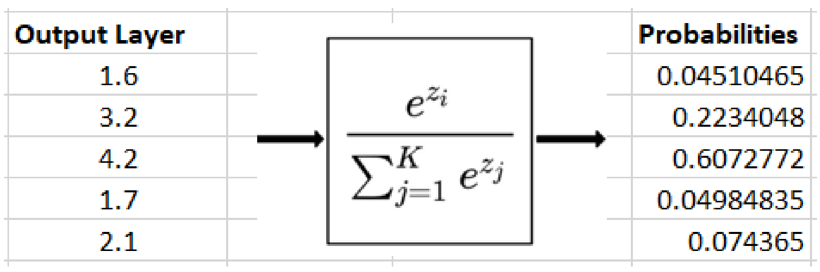
Figure 12 Exponential Linear Unit (ELU) activation function

Figure 12 shows an ELU activation function. When the input is positive, it acts the same as a ReLU activation function. However, it has a small slope for negative inputs. This effectively solves the "dying ReLU" problem.

2.2.1.3 SoftMax Activation Function

Finally, we have the SoftMax activation function. It is commonly used in the output layer for classification problem to generate the probabilities of the prediction being a certain class. It can be written as :

$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



Output Layer		Probabilities
1.6		0.04510465
3.2		0.2234048
4.2	$\xrightarrow{\quad} \boxed{\dfrac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}} \xrightarrow{\quad}$	0.6072772
1.7		0.04984835
2.1		0.074365

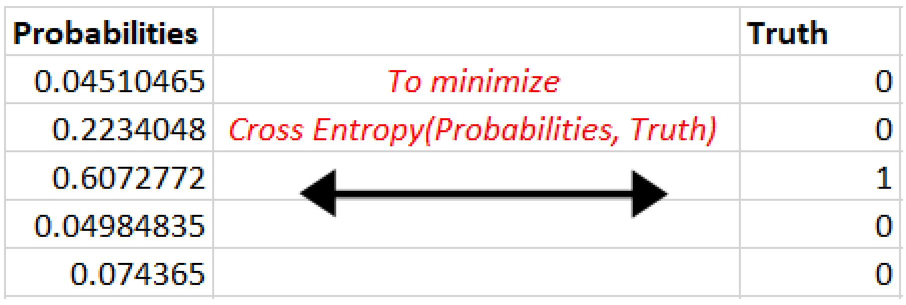
Figure 13 Softmax activation function

2.2.2 Loss functions

2.2.2.1 Cross Entropy Loss Function

Loss functions are used to optimise the model during training. To optimise the model during training, the usual objective is to minimise the loss function. For classification models, the Cross-Entropy loss function is usually used. It is related to the SoftMax activation function discussed earlier.

Recall that in Figure 13, the SoftMax activation function is used to convert logits at the output layer into probabilities. The purpose of the Cross-Entropy loss function is to measure the distance between the truth value and the probabilities shown in Figure 14.



Probabilities		Truth
0.04510465	To minimize	0
0.2234048	Cross Entropy(Probabilities, Truth)	0
0.6072772	$\longleftrightarrow$	1
0.04984835		0
0.074365		0

Figure 14 Aim of Cross Entropy Loss Function

The Cross Entropy Loss Function can be written as:

$$Cross\ Entropy = -\sum_{i=1}^{n} t_i \log{(p_i)}$$

where t_i is the truth label and p_i is the SoftMax probability for the i^{th} class and n is the number of classes.

As shown in Figure 14, the desired output is [0,0,1,0,0] but the model outputs [0.045,0.223,0.607,0.050, 0.074]. The Cross Entropy is therefore calculated as:

$$Cross\ Entropy = -[0 + 0 + \log(0.607) + 0 + 0] = -\log(0.607) = 0.720$$

The objective is to make the model output be as close as possible to the truth values. After a few iterations of training, assume that the model now outputs [0.011,0.110,0.815,0.050,0.014] for the same truth value, the Cross Entropy would now be calculated as:

$$Cross\ Entropy = -[0 + 0 + \log(0.815) + 0 + 0] = -\log(0.815) = 0.294$$

We can thus see that the Cross Entropy Loss decreases with higher accuracy. Hence, minimising the Cross Entropy is the objective of training. We said that the model is learning when the Cross Entropy has been minimised.

2.3 Domain Adversarial Training (DAT)

The Domain Adversarial Training (DAT) of Neural Network [21] is a representation learning approach for domain adaptation, in which data at training and test time come from similar but different distributions. It has achieved state-of-the-art results on a variety of benchmark in domain adaptation, namely for sentiment analysis and image classification task.

Domain adaptation happens when a discriminative classifier is learnt in the presence of a shift between the training and test distribution. Domain Adversarial Training builds mappings between the source domain and target domains, so that the classifier learned with the source domain is also applicable to the target domain. The source and target domain can come from different datasets. We refer to a neural network built using DAT as a Domain Adversarial Neural Network (DANN).

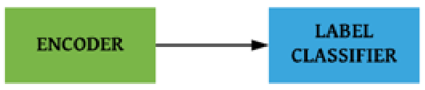
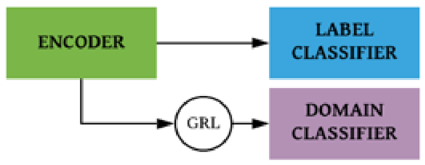
Structural Differences between NN and DANN	
NN	
DANN	

Figure 15 Structural Difference between NN and DANN

Figure 15 shows the simplified structural of a normal neural network (NN) and a DANN. Structurally, the only difference between a normal NN and DANN is the addition of a Gradient Reversal Layer (GRL) and a domain classifier to the latter. The structure of DANN allows us to embed domain adaptation into the process of learning representation so that the label classifications are made based on features that are both discriminative and invariant to the change in domain. During training, the parameters of the underlying deep feature mapping are optimised to minimise the loss of the label classifier and maximise the loss of the domain classifier. The latter is enabled with the use of the GRL that leaves the features unchanged during forward propagation and reverses the gradient by multiplying it by a negative scalar during the backpropagation, thus working in an adversarial manner to the domain classifier and encourages the emergence of domain-invariant features. The loss function of DAT can be defined as:

$$J(\theta) = \sum_{i=1}^{N} (J(e_i, \hat{e}_i : X_i, \Theta) - \lambda\ J(s_i, \hat{s}_i : X_i, \Theta))$$

where Θ , $J(e_i, \hat{e}_i : X_i, \Theta)$ and $J(s_i, \hat{s}_i : X_i, \Theta)$ are the model parameters, loss of label classifier and loss of domain classifier respectively. In our experiment, we called λ as our domain adaptation parameter and it is a dynamic value that increases according with the progress of the training which will make the domain classifier less sensitive to noisy signal at the beginning of the training[17], [18].

$$\lambda = \frac{2}{1 + \exp(-\gamma \cdot p)} - 1$$

Here, p is the training progress and γ is a positive annealing hyper parameter. We have experimented with $\gamma = 1.25, 2.5, 3.33$ in our experiment.

By treating the speakers as domain, and labels as emotions, speaker-invariant SER can also be framed as a domain adaptation problem since the training, validation and test set contains audio instances from different speakers.

DAT has also been investigated in removing speaker variability in speech by learning speaker invariant representations [17], [18]. DAT algorithms for SIER tasks consist of an encoder which learn a speaker invariant representation, an emotion classifier to identify emotion and a speaker classifier to remove speaker variability. DAT removes speaker variability by minimizing emotion classifier cost and maximizing the speaker classifier cost. The maximization is done by reversing the gradient of the speaker classifier.

2.4 Audio Signal Processing

Since we are doing Speaker Invariant Speech Emotion Recognition, we would need to understand what are the inputs we can feed into our model architecture. For training SER system, 2 of the most used features are Log Mel Spectrogram (LMS) and Mel Frequency Cepstral Coefficients (MFCC). We will need to go through some of the technicalities behind Digital Signal Processing to understand how they are formed so that we can used them in our experiments.

2.4.1 Audio and Spectrum Processing

First, we will go through what is an audio signal. An audio signal is a variation in the air pressure over time. To capture the information of the air pressure digitally, we take samples of the air pressure over time. The sampling rate is the number of samples taken over some fixed amount of time. A high sampling rate usually correlates to lesser information loss but heavy computation whereas a low sampling rate has higher information loss but cheaper computation. In this project, the sampling rate of the Emo-DB dataset and RAVDESS Dataset are 16000 Hz and 48000 Hz, respectively. As shown in the Figure 16, an audio signal could be represented as a time-series along the x-axis with the y-axis representing the amplitude of the waveform.

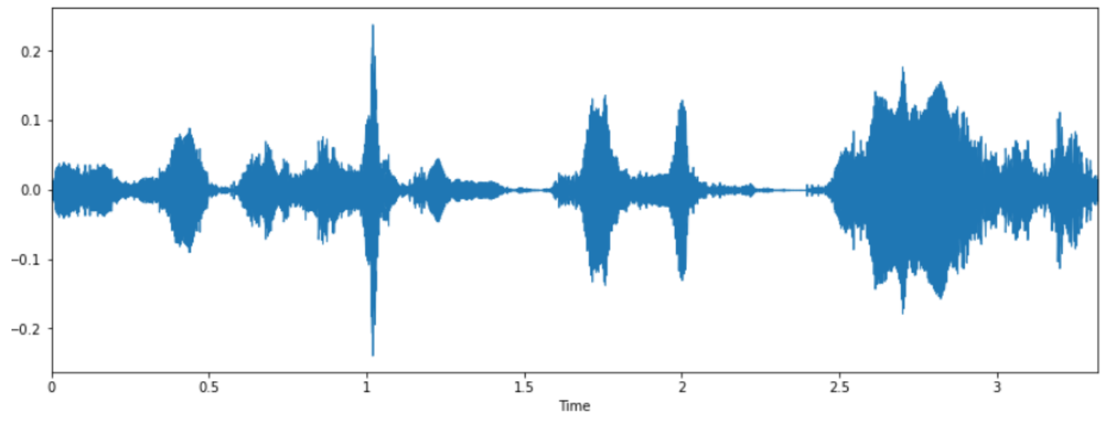


Figure 16 Example of an audio signal from the RAVDESS dataset

An audio signal is made up of several single-frequency sound waves superimposed onto each other. When we take samples of signal over time, we are capturing the resulting amplitudes of the different sound waves. The Fourier Transform can be used to decompose the audio signal into its individual frequencies and the frequency's amplitude. In other words, the signal from the time domain is converted into the frequency domain forming what is known as a spectrum. We can use the Fast Fourier Transform (FFT) algorithm to compute the Fourier Transform as shown in Figure 17.

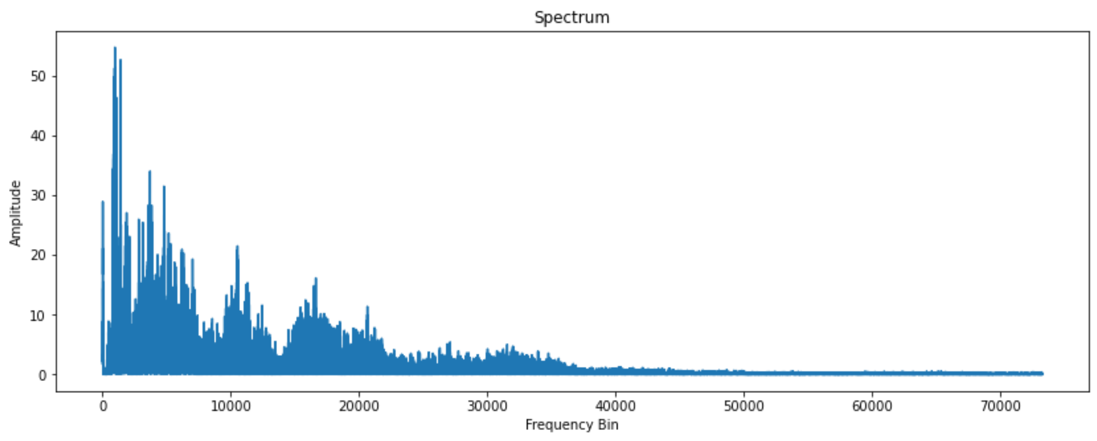


Figure 17 Spectrum of the audio signal computed by Fourier Transform

In speech, the audio signals are non-periodic in nature. This means that the signal's frequency content will vary over time. To represent the spectrum over time, we perform Short-Time Fourier Transform (STFT). Before delving into STFT, we will first look at the concept of audio frames. As mentioned earlier, an audio signal is made up of samples of air pressure over time. For the RAVDESS dataset where the sampling rate is 48000Hz, a single sample point will correspond to 0.0108ms which would be too short a duration to be perceivable by the human ear. To make sure that the audio samples are perceivable, we can combine these sample points into an audio frame. The frame size is thus the number of samples in a frame.

Setting the frame size as the number of sample points in the entire waveform is equivalent to treating the entire audio waveform as a single frame. By performing FFT on the entire waveform as a single frame, we get the spectrum in Figure 17.

In STFT, instead of treating the entire waveform as a single frame and perform FFT on it, we first divided the waveform into many frames. In our experiments, we set the frame size to be 2048.

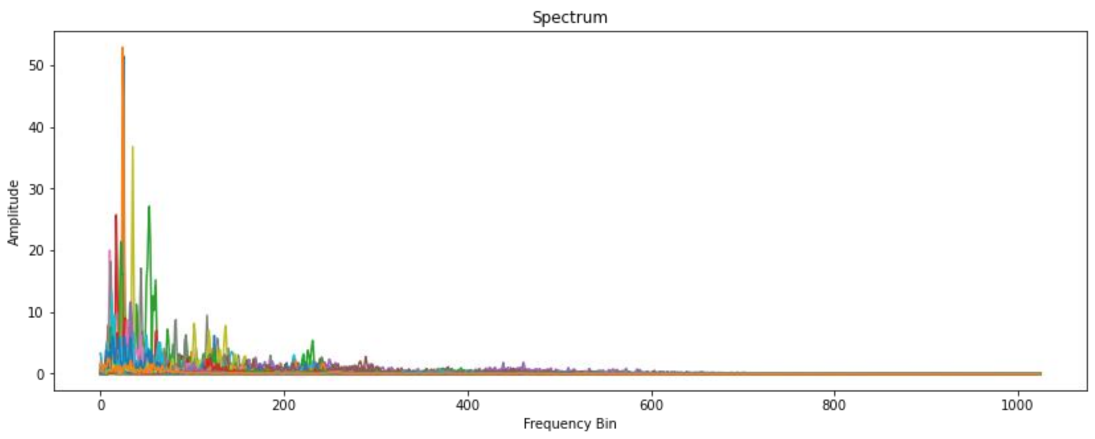


Figure 18 Spectrums of the audio signal computed by STFT with frame size = 2048.

Figure 18 shows the spectrums for the first few frames in the audio waveform, where different colours represent the spectrum for each frame. Due to limited colour allowed in the plot, the figure does not show the spectrums for all the frames. Given that there is no overlapping of frame at this point of the discussion, the total number of frames can be easily calculated as the number of samples divided by the frame size.

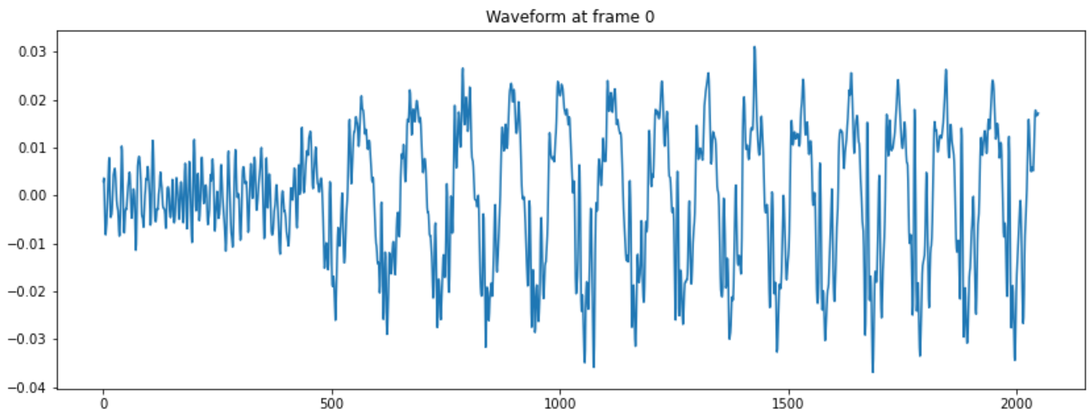


Figure 19 First frame of the audio waveform

Oftentimes, when we are processing a signal with FFT or STFT that does not have an integer number of periods, the endpoints of the signal will be discontinuous. These discontinuities will appear as high frequency components in the spectrum even though they are absent in the original signal. The phenomenon known as spectral leakage. The subsequent peaks at the spectrum in Figure 20 could be a result of spectral leakage.

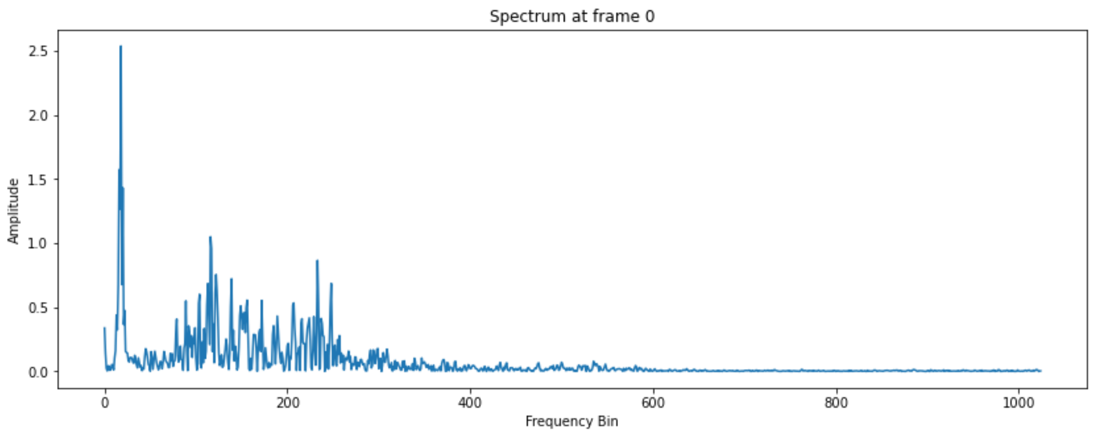


Figure 20 Spectrum of the first frame of the audio waveform

To minimise spectral leakage, we can apply a window function, the Hann Window to each frame of the waveform before feeding them into Fourier transform. In our experiment, the window size is set to be the same as our frame size of 2048. This will eliminate the samples at both ends of a frame as shown the Figure 21.

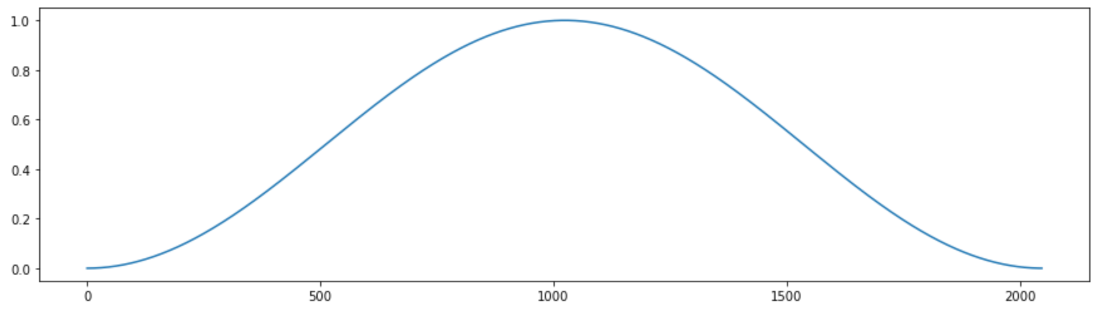


Figure 21 A Hann Window Function

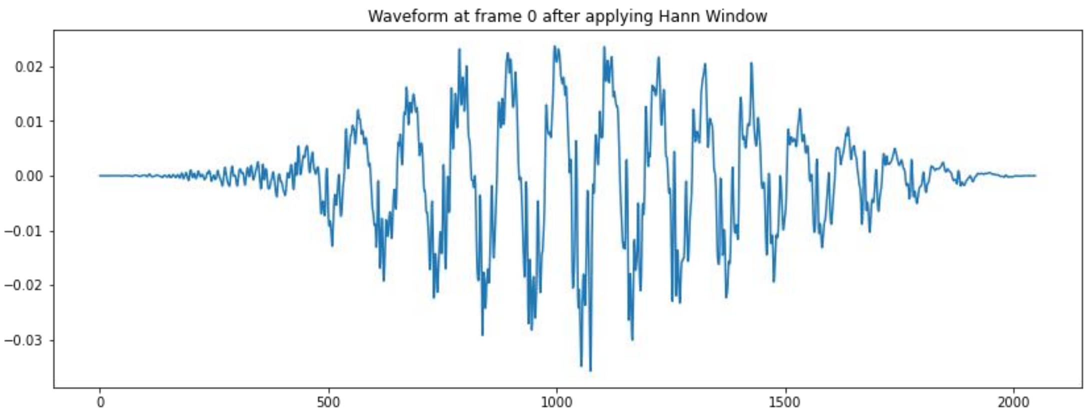


Figure 22 First frame of the audio waveform with Hann Window

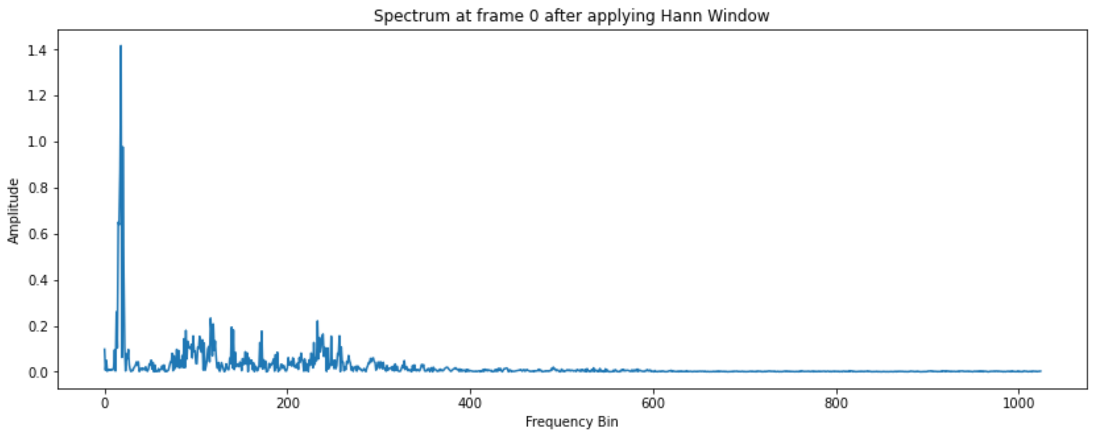


Figure 23 Spectrum of the first frame of the audio waveform with Hann Window

Comparing Figure 19 and Figure 22, we can clearly see how the samples at both ends of a frame can be eliminated via Hann Window. Since the samples at both ends of a frame are eliminated, we mitigate the issue where the endpoints of the signal or frame are discontinuous, thereby minimising spectral leakage.

Comparing Figure 20 and Figure 23, we can see a sharp distinction between the spectrum before and after the Hann Window is applied to the first frame of the audio waveform, proving that Hann Window indeed help in addressing spectral leakage.

While Hann Window is an effective method to minimise spectral leakage, it creates another problem. As observed in Figure 24, both endpoints of every frame are now eliminated. If we were to conjoin the frames back, we will have parts in between the frames where there are no signal. In another word, applying Hann Window to each frame without additional measure will result in information loss.

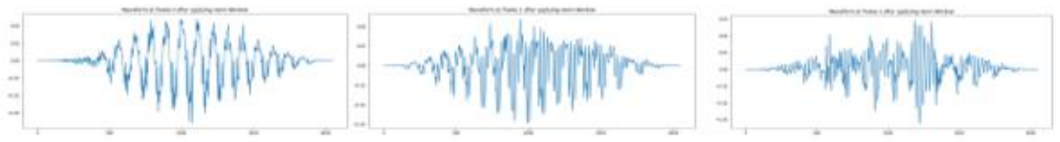


Figure 24 First 3 frames of the audio waveform after applying Hann Window

To overcome this conundrum, we can overlap the frames. In particular, the term hop length is used to describe the number of samples we shift to the right from the starting sample of the current frame to get the next frame. For example, given a frame size of 2048 and hop length of 512, the first frame will start at the 1st sample and end at the 2048th sample while the second frame will start at the 512th sample and end at the 2560th sample. In our experiments, our hop length is set to 512. Figure 25 illustrate the idea of overlapping frames.

2.4.2 Log Mel Spectrogram (LMS)

Having discussed audio signals and spectrums, we will move on to spectrogram. A spectrogram is a visual representation of spectrum of frequencies as it varies with time. Figure 25 shows the entire audio waveform converted to a spectrogram, with frame size and window size of 2048 and hop length of 512. The x-axis represents time, and the y-axis represents frequency converted to logarithmic scale. The color axis (amplitude) is also converted to Decibels which is of a logarithmic scale of amplitudes as well.

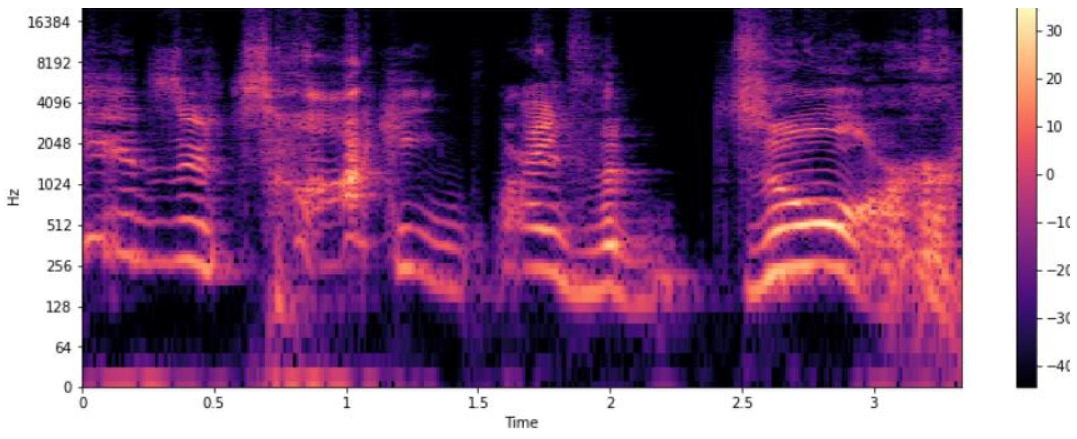


Figure 25 Log spectrogram of the audio waveform

Since human do not perceived frequencies on a linear scale, a unit of pitch was proposed in 1937 such that equal distances in pitch sounded equally distant to the listener. This is known as the Mel-scale. The equation for converting for frequency to Mel-scale is:

$$M(f) = 1125 \ln\left(1 + \frac{f}{700}\right)$$

Figure 26 shows the result after converting the Log-scale spectrogram to Log-Mel-scale spectrogram.

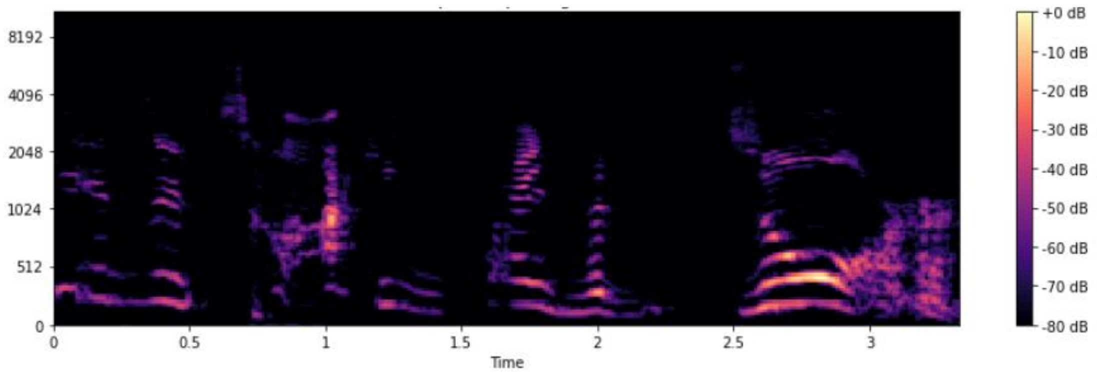


Figure 26 Log-Mel Spectrogram of the audio waveform

2.4.3 Mel Frequency Cepstral Coefficients (MFCC)

Like Spectrogram, Log Spectrogram and Log-Mel Spectrogram, MFCC is also a visual representation of spectrum of frequencies features as it varies with time.

To generate MFCCs, there are also a series of step we can follow. Without going deep into the details, the steps are:

1. Slice the signal into short frames (of time)
2. Compute the periodogram estimate of the power spectrum for each frame
3. Apply the Mel-Filterbank to the power spectra and sum the energy is each filter
4. Take the Discrete Cosine Transform (DCT) of the log Filterbank energies

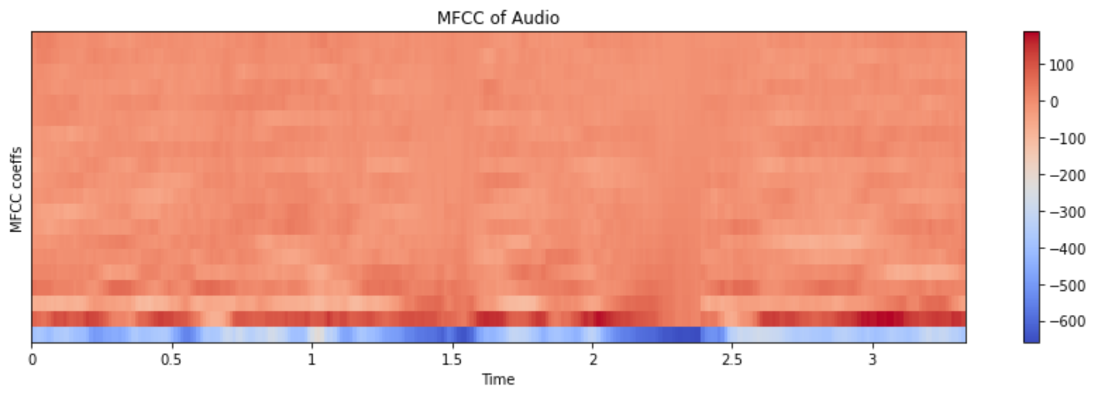


Figure 27 MFCC of the audio waveform

Both the LMS and MFCC will be used as input in our experiments. The 2-dimensional features (frequencies and time) of LMS and MFCC serves as our inspiration to use 2D CNN with RNN in our encoder architecture.

3. Methodology

3.1 Data Pre-processing

This section talks about the pre-processing of audio data. It is best to refer to section 2.4 on Digital Signal Processing to appreciate why certain pre-processing steps are made. In this project, experiments are conducted using 2 different datasets. Since each audio file in each datasets are not of the same length, and that we are not going to use raw audio in our experiments due to the excessive memory requirement for the GPU, data pre-processing is needed.

3.1.1 Removing silence from raw audio

The first step that we do is to remove silence at both the start and the end of the raw audio. These silence do not contain any information and hence is of no use in our task of speaker invariant emotion recognition. We empirically determine silence to be below 25 decibels in our dataset and trimmed them away. This can be done with the *Librosa* package with just one line.

```
trimmed_signal, _ = librosa.effects.trim(signal,top_db = 25)
```

Figure 28 shows the effect of trimming of one of the audio waveform in the dataset.

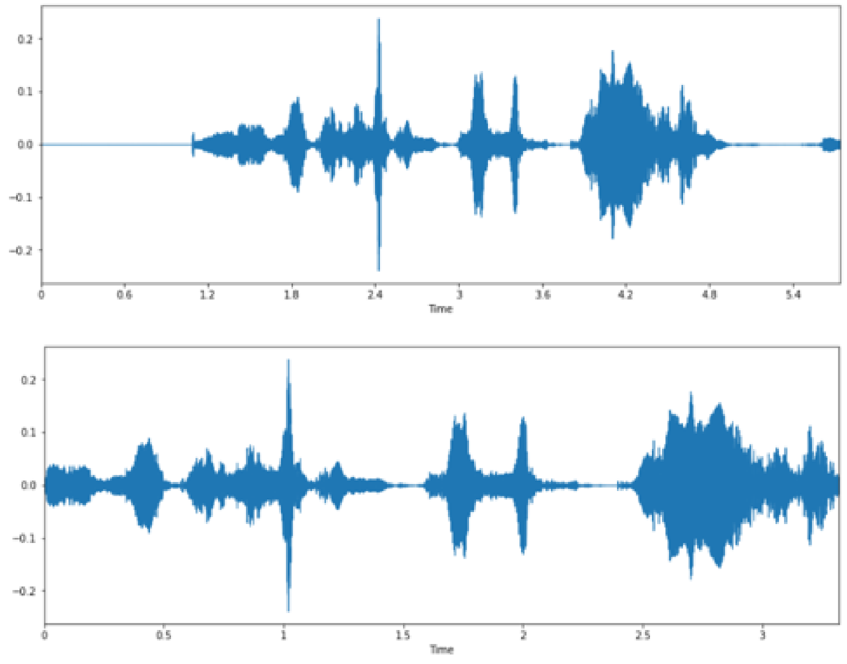


Figure 28 Trimming of audio signal to remove silence

3.1.2 Removing background noise from audio

After listening to the trimmed audio files, we realised that there is some background noise with some of them . Hence, we employ the use Wiener filtering which aims to remove the background noise. Wiener filters help to remove background noise by subtracting an estimation of the noise spectrum from the noisy speech spectrum [22]. In this project, this is done with the help of the *SciPy* package with just one line.

```
signal_wiener = scipy.signal.wiener(trimmed_signal)
```

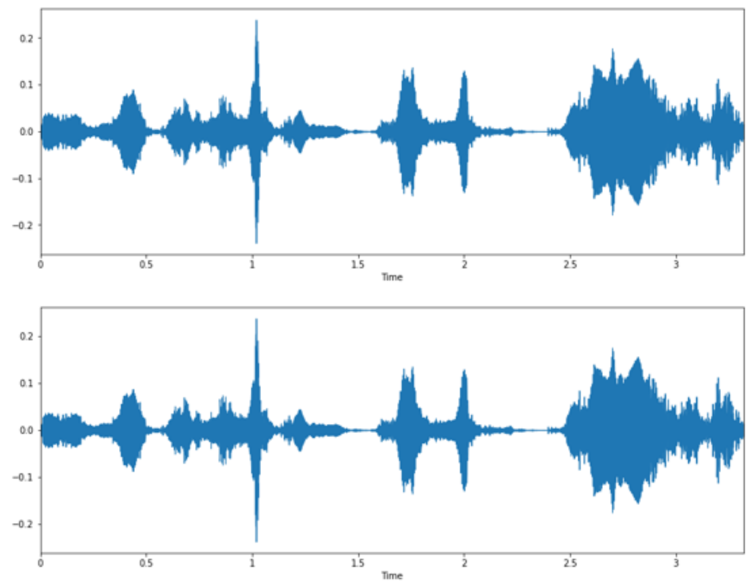


Figure 29 Applying Wiener filtering to remove background noise

Figure 29 shows the audio signal before and after applying Wiener filtering. It does not seem to have any differences when we first see it. But we can calculate the noise difference to see if Wiener filtering helps at all.

```
diff_noise = signal_wiener - trimmed_signal
```

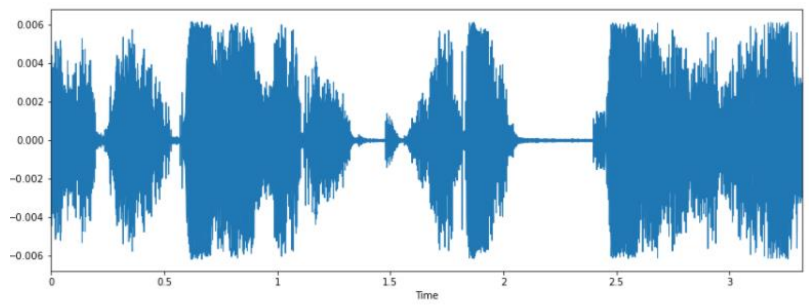


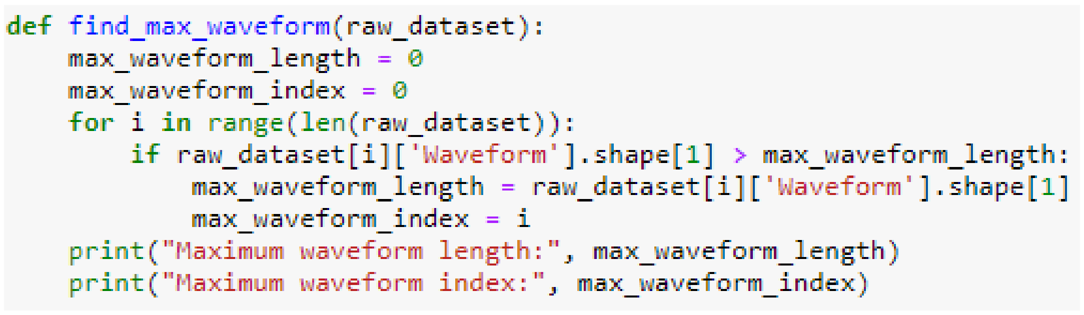
Figure 30 The background noise difference

Figure 30 shows that there is indeed some background noise difference that Wiener filtering has help to remove.

3.1.3 Padding audio to the longest audio length

Since we would be doing batch training, and different audio recordings in the dataset are of different length after trimming and background noise removal, we iterate through all the audio recordings in the dataset and padded them with 0, so they are of the same length as the longest audio recordings in the dataset.

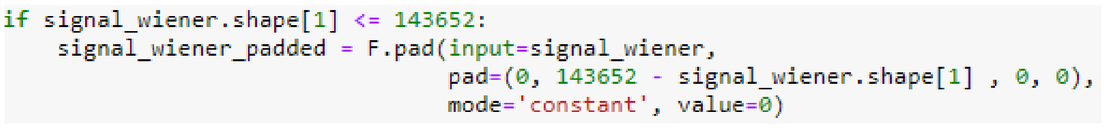
The maximum waveform length and its index can be found as demonstrated in Code Snippet 1.



```python
def find_max_waveform(raw_dataset):
    max_waveform_length = 0
    max_waveform_index = 0
    for i in range(len(raw_dataset)):
        if raw_dataset[i]['Waveform'].shape[1] > max_waveform_length:
            max_waveform_length = raw_dataset[i]['Waveform'].shape[1]
            max_waveform_index = i
    print("Maximum waveform length:", max_waveform_length)
    print("Maximum waveform index:", max_waveform_index)
```

Code Snippet 1 Finding maximum audio length and index in dataset

Assuming that the maximum waveform length in the dataset is 143652, any waveform that is shorter than 143652 can then be padded with 0 so that it is of 143562 length as shown in Code Snippet 2.

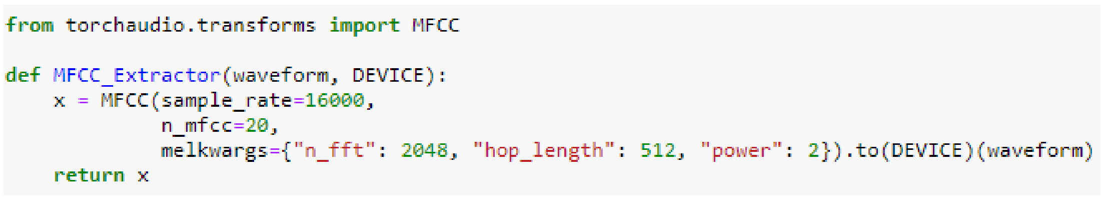


```python
if signal_wiener.shape[1] <= 143652:
    signal_wiener_padded = F.pad(input=signal_wiener,
                                 pad=(0, 143652 - signal_wiener.shape[1] , 0, 0),
                                 mode='constant', value=0)
```

Code Snippet 2 Padding audio with 0 to the maximum audio length in dataset

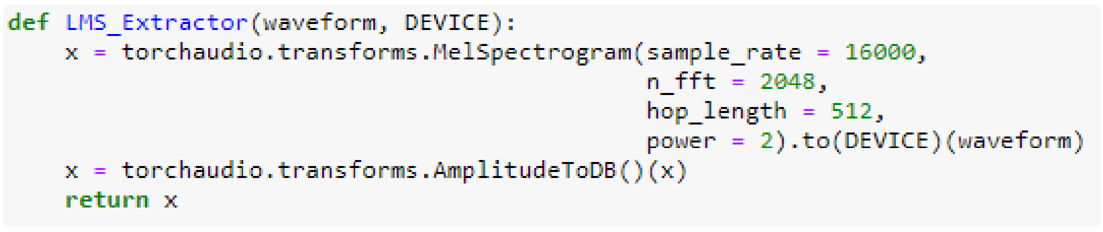
3.1.4 Conversion of pre-processed audio to low-level features

To convert the pre-processed audio to low level features is easy. In our case, we used the *torchaudio* library to convert the audio to low level features as shown in Code Snippet 3, 4 and 5. The low-level features in Code Snippet 5 is used for in the experiments for an existing DAT encoder.



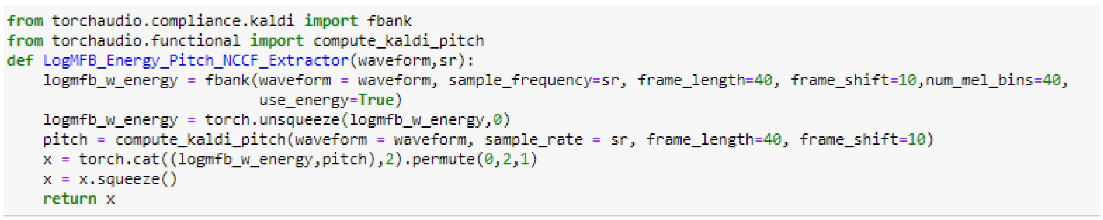
```python
from torchaudio.transforms import MFCC

def MFCC_Extractor(waveform, DEVICE):
    x = MFCC(sample_rate=16000,
             n_mfcc=20,
             melkwargs={"n_fft": 2048, "hop_length": 512, "power": 2}).to(DEVICE)(waveform)
    return x
```

Code Snippet 3 Conversion of audio to MFCC



```python
def LMS_Extractor(waveform, DEVICE):
    x = torchaudio.transforms.MelSpectrogram(sample_rate = 16000,
                                             n_fft = 2048,
                                             hop_length = 512,
                                             power = 2).to(DEVICE)(waveform)
    x = torchaudio.transforms.AmplitudeToDB()(x)
    return x
```

Code Snippet 4 Conversion of audio to LMS

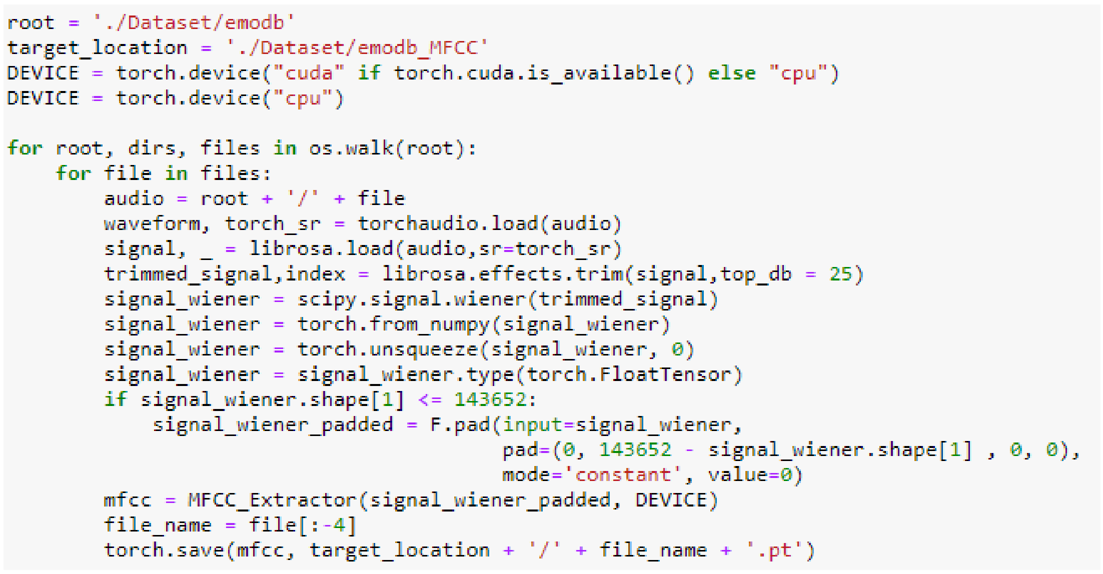


```python
from torchaudio.compliance.kaldi import fbank
from torchaudio.functional import compute_kaldi_pitch
def LogMFB_Energy_Pitch_NCCF_Extractor(waveform,sr):
    logmfb_w_energy = fbank(waveform = waveform, sample_frequency=sr, frame_length=40, frame_shift=10,num_mel_bins=40,
                            use_energy=True)
    logmfb_w_energy = torch.unsqueeze(logmfb_w_energy,0)
    pitch = compute_kaldi_pitch(waveform = waveform, sample_rate = sr, frame_length=40, frame_shift=10)
    x = torch.cat((logmfb_w_energy,pitch),2).permute(0,2,1)
    x = x.squeeze()
    return x
```

Code Snippet 5 Conversion of audio to LogMFB + Energy + Pitch with NCCF

We have experimented with various window size, frame size and hop length and in the end we have decided to use a frame size and window size of 2048 and a hop length of 512 for both MFCC and LMS. The window function is set to Hann Window by default in *torchaudio.* For MFCC, we only retain the first 20 coefficients.

Combining all of them together, we get Code Snippet 5.



```python
root = './Dataset/emodb'
target_location = './Dataset/emodb_MFCC'
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
DEVICE = torch.device("cpu")

for root, dirs, files in os.walk(root):
    for file in files:
        audio = root + '/' + file
        waveform, torch_sr = torchaudio.load(audio)
        signal, _ = librosa.load(audio,sr=torch_sr)
        trimmed_signal,index = librosa.effects.trim(signal,top_db = 25)
        signal_wiener = scipy.signal.wiener(trimmed_signal)
        signal_wiener = torch.from_numpy(signal_wiener)
        signal_wiener = torch.unsqueeze(signal_wiener, 0)
        signal_wiener = signal_wiener.type(torch.FloatTensor)
        if signal_wiener.shape[1] <= 143652:
            signal_wiener_padded = F.pad(input=signal_wiener,
                                         pad=(0, 143652 - signal_wiener.shape[1] , 0, 0),
                                         mode='constant', value=0)
        mfcc = MFCC_Extractor(signal_wiener_padded, DEVICE)
        file_name = file[:-4]
        torch.save(mfcc, target_location + '/' + file_name + '.pt')
```

Code Snippet 6 Trimming, filtering, padding of audio before conversion

Code Snippet 5 will go through all the audio files in the dataset to trim them, and filter them, pad to the longest audio length, and convert them to low level features before saving them in another directory.

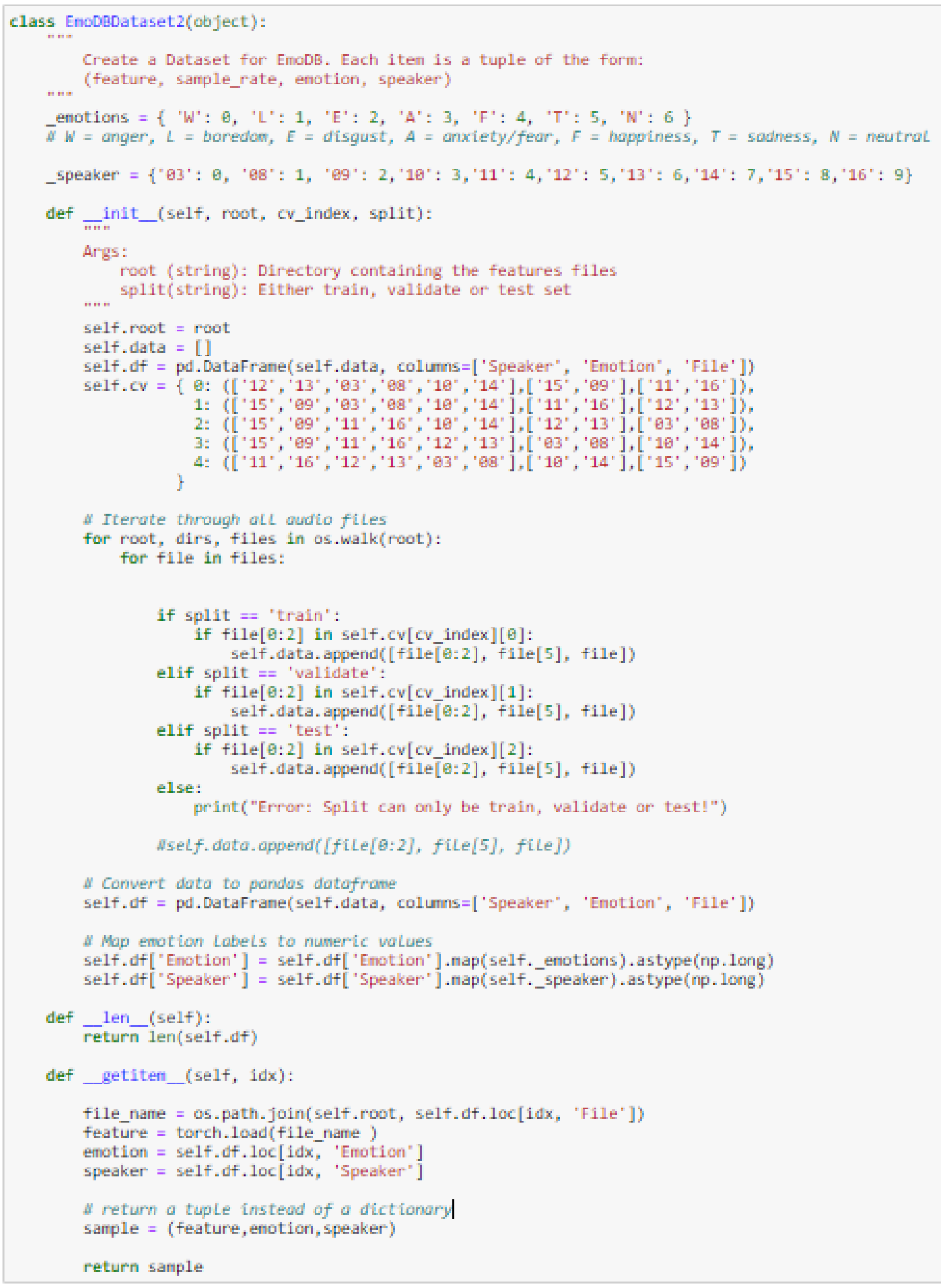
3.2 Preparing pre-processed data for K Fold Cross-Validation

The 2 datasets we used in our experiments are:

1. A database of German Emotional Speech (Emo-DB)[19]
2. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)[20]

3.2.1 Emo-DB Dataset

Emo-DB dataset contains speech recording of 7 emotions representing anger, boredom, disgust, fear, happy, sad and neutral, from 10 German speaking persons. There were total of 535 speech samples. For speaker invariant SER, we performed 5-fold leave-two-speakers-out for cross validation and testing. That is, in each fold 6 speakers were selected for training, 2 speakers for validation and 2 speakers for testing. The ratio of male to female speakers in training, validation, and testing was set to 1:1. We use the custom dataset class to achieve this purpose. This is shown in Code Snippet 6.

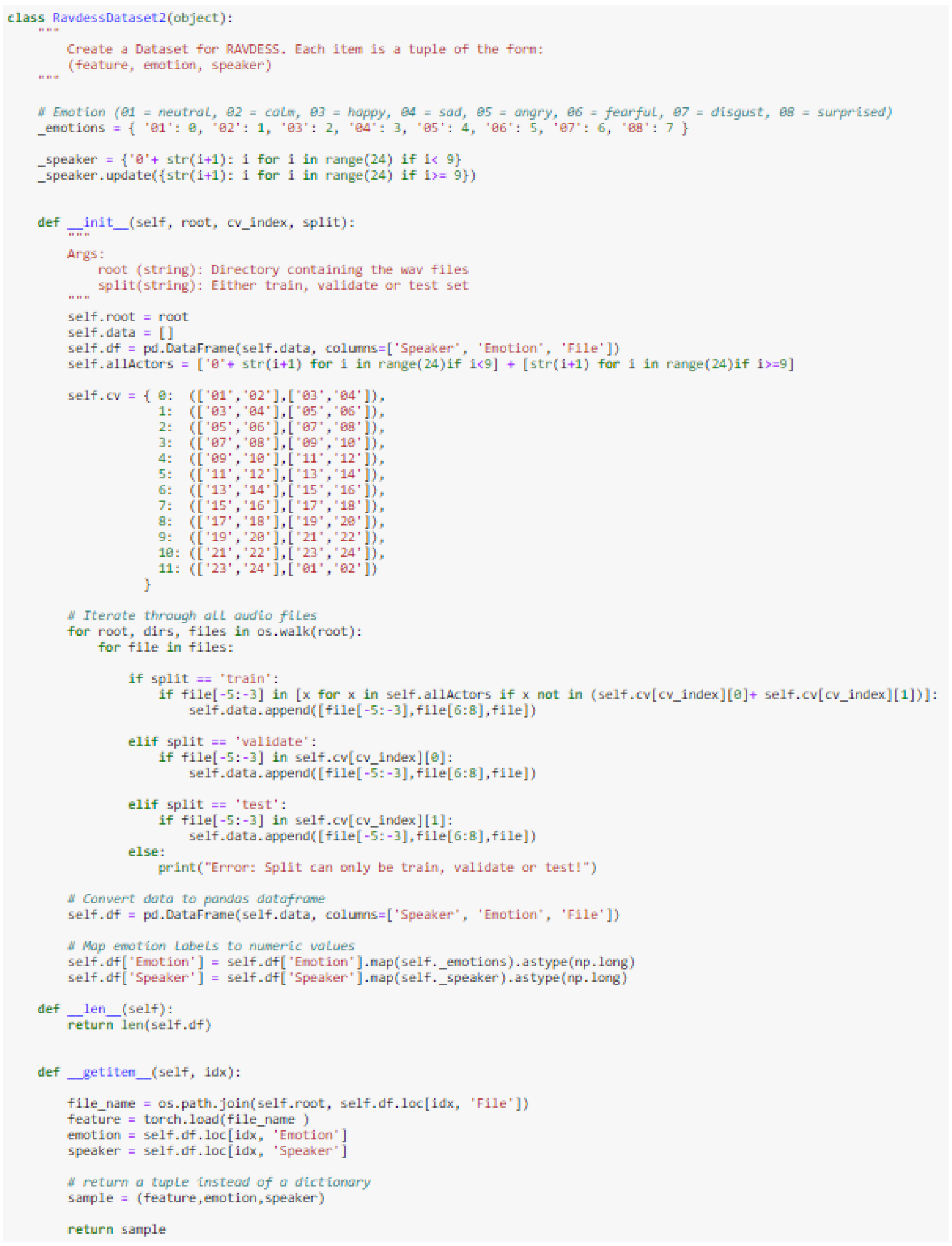


```python
class EmoDBDataset2(object):
    """
        Create a Dataset for EmoDB. Each item is a tuple of the form:
        (feature, sample_rate, emotion, speaker)
    """
    _emotions = { 'W': 0, 'L': 1, 'E': 2, 'A': 3, 'F': 4, 'T': 5, 'N': 6 }
    # W = anger, L = boredom, E = disgust, A = anxiety/fear, F = happiness, T = sadness, N = neutral

    _speaker = {'03': 0, '08': 1, '09': 2,'10': 3,'11': 4,'12': 5,'13': 6,'14': 7,'15': 8,'16': 9}

    def __init__(self, root, cv_index, split):
        """
        Args:
            root (string): Directory containing the features files
            split(string): Either train, validate or test set
        """
        self.root = root
        self.data = []
        self.df = pd.DataFrame(self.data, columns=['Speaker', 'Emotion', 'File'])
        self.cv = { 0: (['12','13','03','08','10','14'],['15','09'],['11','16']),
                    1: (['15','09','03','08','10','14'],['11','16'],['12','13']),
                    2: (['15','09','11','16','10','14'],['12','13'],['03','08']),
                    3: (['15','09','11','16','12','13'],['03','08'],['10','14']),
                    4: (['11','16','12','13','03','08'],['10','14'],['15','09'])
                  }

        # Iterate through all audio files
        for root, dirs, files in os.walk(root):
            for file in files:


                if split == 'train':
                    if file[0:2] in self.cv[cv_index][0]:
                        self.data.append([file[0:2], file[5], file])
                elif split == 'validate':
                    if file[0:2] in self.cv[cv_index][1]:
                        self.data.append([file[0:2], file[5], file])
                elif split == 'test':
                    if file[0:2] in self.cv[cv_index][2]:
                        self.data.append([file[0:2], file[5], file])
                else:
                    print("Error: Split can only be train, validate or test!")

                #self.data.append([file[0:2], file[5], file])

        # Convert data to pandas dataframe
        self.df = pd.DataFrame(self.data, columns=['Speaker', 'Emotion', 'File'])

        # Map emotion labels to numeric values
        self.df['Emotion'] = self.df['Emotion'].map(self._emotions).astype(np.long)
        self.df['Speaker'] = self.df['Speaker'].map(self._speaker).astype(np.long)

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):

        file_name = os.path.join(self.root, self.df.loc[idx, 'File'])
        feature = torch.load(file_name )
        emotion = self.df.loc[idx, 'Emotion']
        speaker = self.df.loc[idx, 'Speaker']

        # return a tuple instead of a dictionary
        sample = (feature,emotion,speaker)

        return sample
```

Code Snippet 7 Emo-DB dataset for 5-fold-leave-2-speaker out cross validation

3.2.2 RAVDESS Dataset

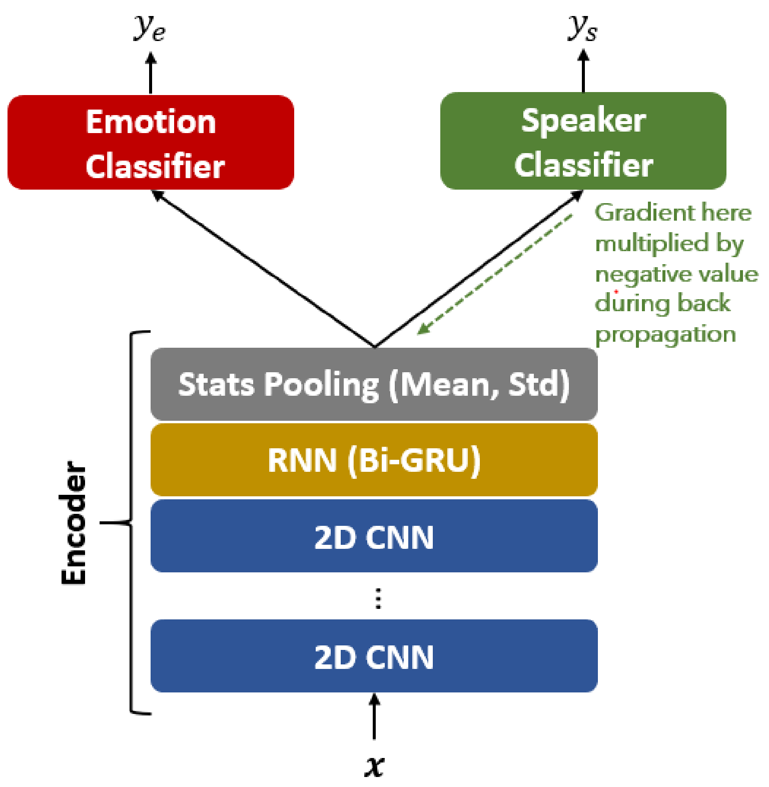
RAVDESS dataset contains speech recordings of 8 emotions representing anger, calm, surprised, fear, happy, sad, neutral and disgust, from 24 professional actors with north American English accent. There were total of 1440 speech samples. We performed 12-fold leave-two-speakers-out cross validation by selecting 20 speakers for training, 2 for validation and 2 for testing. The ratio of male to female speakers in training, validation and testing was set to 1:1. We use the custom dataset class to achieve this purpose. This is shown in Code Snippet 7.



```python
class RavdessDataset2(object):
    """
        Create a Dataset for RAVDESS. Each item is a tuple of the form:
        (feature, emotion, speaker)
    """

    # Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised)
    _emotions = { '01': 0, '02': 1, '03': 2, '04': 3, '05': 4, '06': 5, '07': 6, '08': 7 }

    _speaker = {'0'+ str(i+1): i for i in range(24) if i< 9}
    _speaker.update({str(i+1): i for i in range(24) if i>= 9})


    def __init__(self, root, cv_index, split):
        """
        Args:
            root (string): Directory containing the wav files
            split(string): Either train, validate or test set
        """
        self.root = root
        self.data = []
        self.df = pd.DataFrame(self.data, columns=['Speaker', 'Emotion', 'File'])
        self.allActors = ['0'+ str(i+1) for i in range(24)if i<9] + [str(i+1) for i in range(24)if i>=9]

        self.cv = { 0:  (['01','02'],['03','04']),
                    1:  (['03','04'],['05','06']),
                    2:  (['05','06'],['07','08']),
                    3:  (['07','08'],['09','10']),
                    4:  (['09','10'],['11','12']),
                    5:  (['11','12'],['13','14']),
                    6:  (['13','14'],['15','16']),
                    7:  (['15','16'],['17','18']),
                    8:  (['17','18'],['19','20']),
                    9:  (['19','20'],['21','22']),
                    10: (['21','22'],['23','24']),
                    11: (['23','24'],['01','02'])
                  }

        # Iterate through all audio files
        for root, dirs, files in os.walk(root):
            for file in files:

                if split == 'train':
                    if file[-5:-3] in [x for x in self.allActors if x not in (self.cv[cv_index][0]+ self.cv[cv_index][1])]:
                        self.data.append([file[-5:-3],file[6:8],file])

                elif split == 'validate':
                    if file[-5:-3] in self.cv[cv_index][0]:
                        self.data.append([file[-5:-3],file[6:8],file])

                elif split == 'test':
                    if file[-5:-3] in self.cv[cv_index][1]:
                        self.data.append([file[-5:-3],file[6:8],file])
                else:
                    print("Error: Split can only be train, validate or test!")

        # Convert data to pandas dataframe
        self.df = pd.DataFrame(self.data, columns=['Speaker', 'Emotion', 'File'])

        # Map emotion labels to numeric values
        self.df['Emotion'] = self.df['Emotion'].map(self._emotions).astype(np.long)
        self.df['Speaker'] = self.df['Speaker'].map(self._speaker).astype(np.long)

    def __len__(self):
        return len(self.df)


    def __getitem__(self, idx):

        file_name = os.path.join(self.root, self.df.loc[idx, 'File'])
        feature = torch.load(file_name )
        emotion = self.df.loc[idx, 'Emotion']
        speaker = self.df.loc[idx, 'Speaker']

        # return a tuple instead of a dictionary
        sample = (feature,emotion,speaker)

        return sample
```

Code Snippet 8 RAVDESS dataset for 12-fold-leave-2-speaker out cross validation

3.3 DAT Network Architecture



y_e
y_s
Emotion Classifier
Speaker Classifier
Gradient here multiplied by negative value during back propagation
Stats Pooling (Mean, Std)
RNN (Bi-GRU)
2D CNN
2D CNN
Encoder
x

Figure 31 Our proposed DAT network architecture

Our proposed DAT network architecture consists of 3 components: (i) an encoder which is made from one to multiple layers of 2D CNN(s) , a Bidirectional GRU (biGRU) and a statistical pooling layer, which generates speaker invariant representations; (ii) an emotion classifier that predicts emotional labels; and (iii) a speaker classifier which removes speaker variability.

3.3.1 Encoder

The network takes LMS or MFCC features of speech in a time window size of T . Input speech emotion data $x = \big(x(t,n)\big)$ where $x \in \mathbb{R}^{T\times n}$ and n denotes the number of features. The encoder consists of a one or more 2D CNN layers followed by one RNN layer. The 2D CNN aims to learn short-term frequency features as well as temporal information as speech contain multiple frequencies. 2D CNN consist of (i) a 2D convolutional layer; (ii) a batch normalization layer; and (iii) a max pooling layer.

Let w_k^1 denote the filter weight connected to k the feature map at the convolutional layer. The output h_k^1 of the k the feature is given by

$$h_k^1 = x \circledast w_k^1$$

where $\circledast$ denote the convolution operation. The convolution layer output is processed by a batch-normalization layer, which is processed by dropout and exponential linear units (eLU). The batch-normalization layer output h_k^2 for k feature map is given by

$$h_k^2 = \mathrm{eLU}\Big(\mathtt{dropout}(w_k^2 \cdot \mathrm{BN}(h_k^1) + b_k^2)\Big)$$

where $\cdot$ denotes the element-wise multiplication, BN denotes the function that normalizes the data by subtracting the mean and dividing by the standard deviation over a batch. w_k^2 and b_k^2 denote learnable parameters. dropouts and eLU denote dropout operation and eLU activation function. Max-pooling layer output h_k^2 is given by

$$h_k^3 = \mathtt{pool}(h_k^2)$$

Let $h_k^3 \in \mathbb{R}^{T \times n^3}$ Where n_3 is the 2D CNN output features.

The RNN learn the long-term temporal relationship of the short-term frequency and temporal features and in this work, we used a Bi-directional Gated Recurrent Unit (biGRU) as the RNN. To achieve this, the convolution layer output $h^3 = (h_k^3) \in \mathbb{R}^{K \times T_c \times n^3}$ is reshaped as $h^3(t) \in \mathbb{R}^{K \times n^3}$. The RNN output $h^4(t)$ is given by:

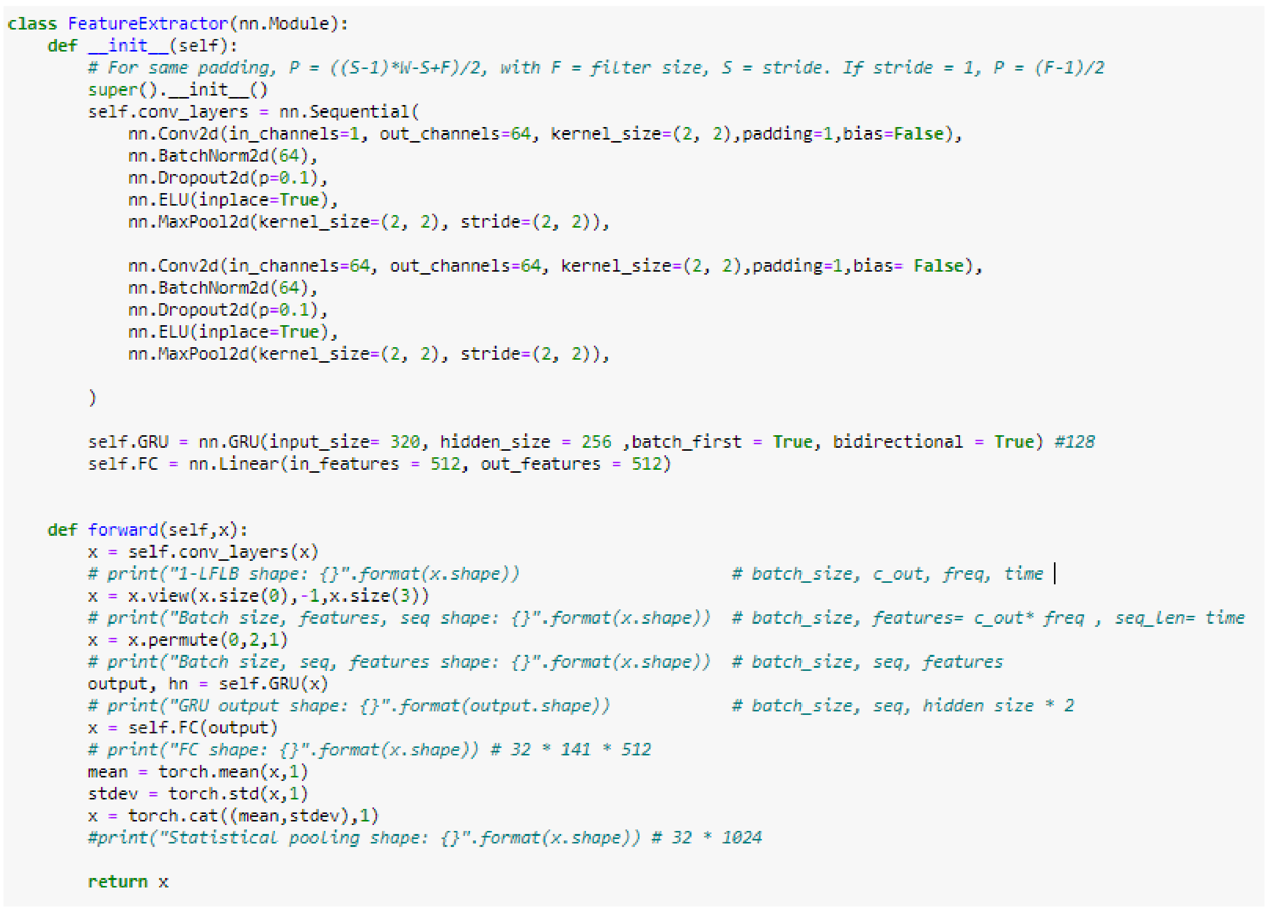
$$h^4(t) = \mathtt{biGRU}\big(h^3(t), h^4(t-1)\big)$$

where $h^4(t) \in \mathbb{R}^{n^4}$ and n^4 is the number of hidden neurons in bi-GRU. The ouput of the stats pooling layer $h^5(t) \in \mathbb{R}^{2*\,n^4}$ is given by:

$$h^5(t) = \mathrm{stats\_pool}(h^4)$$

where stats_pool function calculates the mean and standard deviation of h^4 along the time dimensions and concatenates along the feature dimension.

Code Snippet 8 shows how a 2-layers-2D-CNN with biGRU encoder is created.



```
class FeatureExtractor(nn.Module):
    def __init__(self):
        # For same padding, P = ((S-1)*W-S+F)/2, with F = filter size, S = stride. If stride = 1, P = (F-1)/2
        super().__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=64, kernel_size=(2, 2),padding=1,bias=False),
            nn.BatchNorm2d(64),
            nn.Dropout2d(p=0.1),
            nn.ELU(inplace=True),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)),

            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=(2, 2),padding=1,bias= False),
            nn.BatchNorm2d(64),
            nn.Dropout2d(p=0.1),
            nn.ELU(inplace=True),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)),

        )

        self.GRU = nn.GRU(input_size= 320, hidden_size = 256 ,batch_first = True, bidirectional = True) #128
        self.FC = nn.Linear(in_features = 512, out_features = 512)


    def forward(self,x):
        x = self.conv_layers(x)
        # print("1-LFLB shape: {}".format(x.shape))                    # batch_size, c_out, freq, time |
        x = x.view(x.size(0),-1,x.size(3))
        # print("Batch size, features, seq shape: {}".format(x.shape))  # batch_size, features= c_out* freq , seq_len= time
        x = x.permute(0,2,1)
        # print("Batch size, seq, features shape: {}".format(x.shape))  # batch_size, seq, features
        output, hn = self.GRU(x)
        # print("GRU output shape: {}".format(output.shape))            # batch_size, seq, hidden size * 2
        x = self.FC(output)
        # print("FC shape: {}".format(x.shape)) # 32 * 141 * 512
        mean = torch.mean(x,1)
        stdev = torch.std(x,1)
        x = torch.cat((mean,stdev),1)
        #print("Statistical pooling shape: {}".format(x.shape)) # 32 * 1024

        return x
```

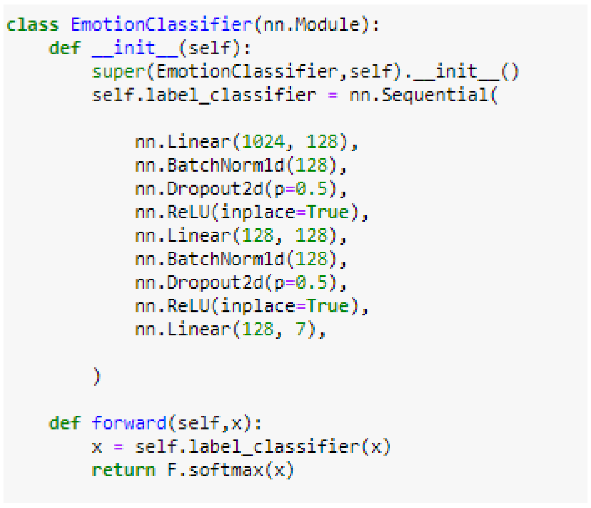
Code Snippet 9 Creation of a 2-layers-2D CNN biGRU encoder

3.3.2 Emotion Classifier

The emotion classifier is used for predicting the emotion labels y_e . It consists of two fully connected layers and each layer performs batch-normalization and dropout before it gets processed by the ReLU activation function. The output layer is a SoftMax layer. It can be written as:

$$\begin{aligned} h^6 &= \mathrm{ReLU}\big(\mathtt{dropout}(W^6\mathrm{BN}(V^6h^5(T)+c^6)+b^6)\big) \\ h^7 &= \mathrm{ReLU}\big(\mathtt{dropout}(W^7\mathrm{BN}(V^7h^6+c^7)+b^7)\big) \\ y_e &= \mathtt{softmax}(W^8h^7+b^8) \end{aligned}$$

Code Snippet 9 shows how the emotion classifier is created for the Emo-DB usage. For RAVDESS, just change the last layer from *nn.Linear(128,7)* to *nn.Linear(128,8).*



```
class EmotionClassifier(nn.Module):
    def __init__(self):
        super(EmotionClassifier,self).__init__()
        self.label_classifier = nn.Sequential(

            nn.Linear(1024, 128),
            nn.BatchNorm1d(128),
            nn.Dropout2d(p=0.5),
            nn.ReLU(inplace=True),
            nn.Linear(128, 128),
            nn.BatchNorm1d(128),
            nn.Dropout2d(p=0.5),
            nn.ReLU(inplace=True),
            nn.Linear(128, 7),

        )

    def forward(self,x):
        x = self.label_classifier(x)
        return F.softmax(x)
```

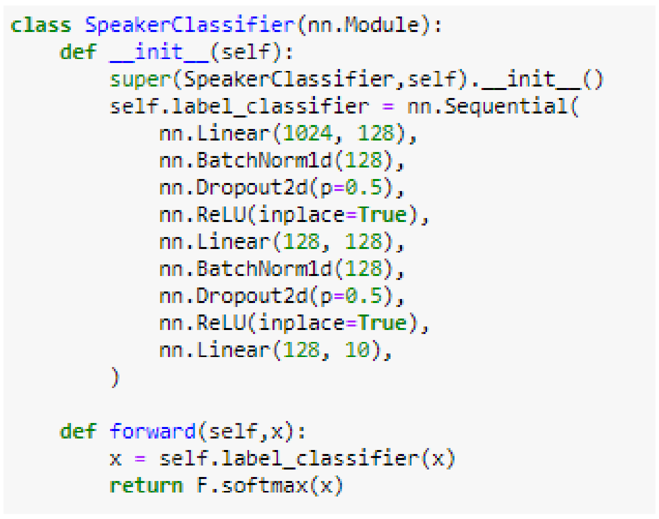
Code Snippet 10 Creation of an Emotion Classifier

3.3.3 Speaker Classifier

The speaker classifier is used for predicting the speaker labels y_s . It consists of two fully connected layers and each layer performs batch-normalization and dropout before it gets processed by the ReLU activation function. The output layer is a SoftMax layer. It can be written as:

$$\begin{aligned} h^9 &= \mathtt{relu}\big(\mathtt{dropout}(W^9 \mathrm{BN}(V^9 h^5(T) + c^9) + b^9)\big) \\ h^{10} &= \mathtt{relu}\big(\mathtt{dropout}(W^{10} \mathrm{BN}(V^{10} h^9 + c^{10}) + b^{10})\big) \\ y_s &= \mathtt{softmax}(W^{11} h^{10} + b^{11}) \end{aligned}$$

Code Snippet 10 shows how the speaker classifier is created for the Emo-DB usage. For RAVDESS, just change the last layer from *nn.Linear(128,10)* to *nn.Linear(128,24).*



```
class SpeakerClassifier(nn.Module):
    def __init__(self):
        super(SpeakerClassifier,self).__init__()
        self.label_classifier = nn.Sequential(
            nn.Linear(1024, 128),
            nn.BatchNorm1d(128),
            nn.Dropout2d(p=0.5),
            nn.ReLU(inplace=True),
            nn.Linear(128, 128),
            nn.BatchNorm1d(128),
            nn.Dropout2d(p=0.5),
            nn.ReLU(inplace=True),
            nn.Linear(128, 10),
        )

    def forward(self,x):
        x = self.label_classifier(x)
        return F.softmax(x)
```

Code Snippet 11 Creation of a Speaker Classifier

3.4 Domain Adversarial Training

We minimize the Cross Entropy loss of the emotion classifier as:

$$J_e = -E_x[d_e log(y_e)]$$

where d_e is the emotion labels and E_x is the expectation over data x.

$$J_s = -E_x[d_s log(y_s)]$$

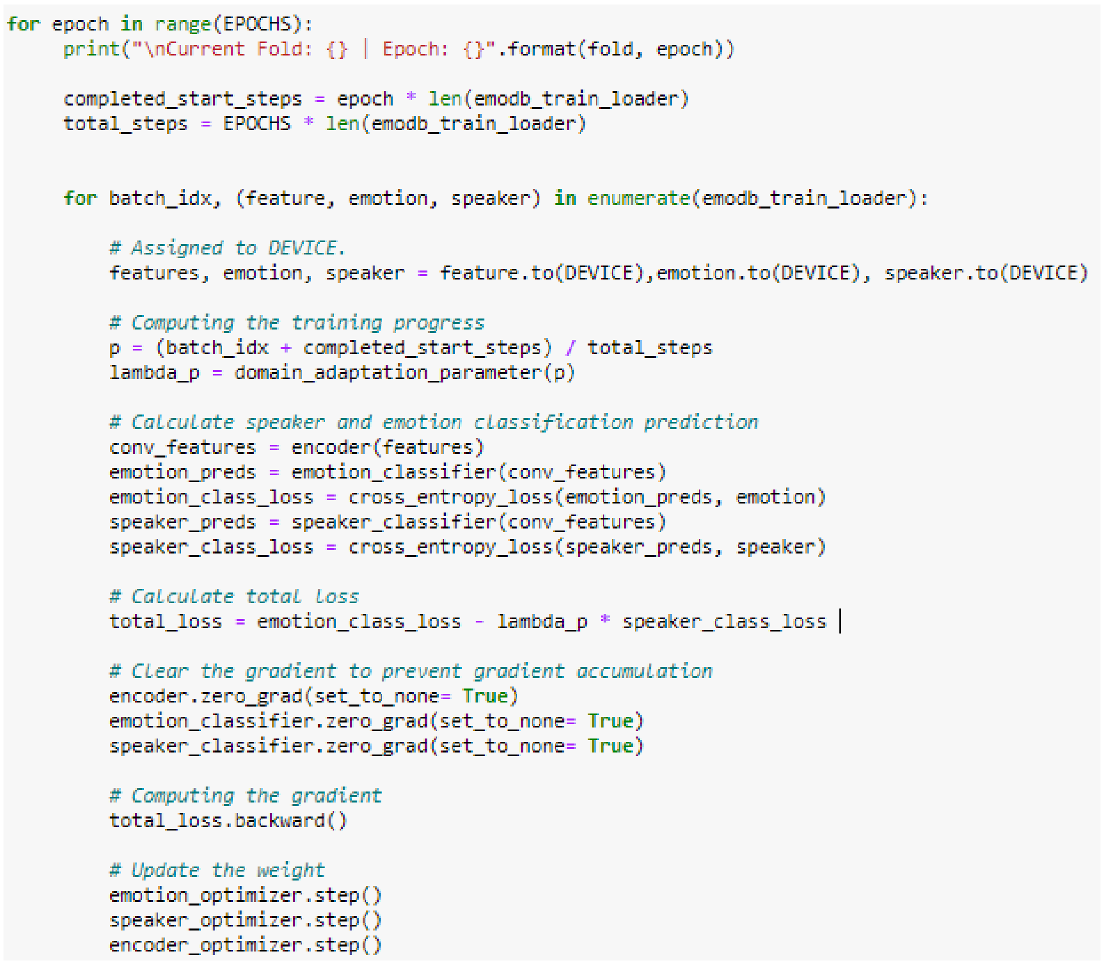
where d_s are the speaker identity labels.

During training, the parameters of the underlying deep feature mapping are optimized to minimize the loss of the emotion classifier J_e and maximise the loss of the speaker classifier J_s. The latter is enabled with the use of the gradient reversal layer that leaves the features unchanged during forward propagation and reverses the gradient by multiplying it by a negative scalar λ during the backpropagation, thus working in an adversarial manner to the speaker classifier and encourages the emergence of speaker-invariant features. Total loss is defined as:

$$J = J_e - \lambda J_s$$

where $\lambda = \frac{2}{1+exp(-\gamma p)} - 1$, where γ is a positive annealing hyper parameter and p is the percentage of training. Importance factor of the speaker classifier λ gradually increases from 0 to at most 1 with the training progress since the negative gradient can hamper the initial weight updates. The rate and degree of speaker adversarial training occurs can be adjusted by changing γ, where a higher γ value will result in a higher rate and degree of speaker adversarial training.

Code Snippet 11 below show the main part of the code responsible for Speaker Adversarial Training



```
for epoch in range(EPOCHS):
    print("\nCurrent Fold: {} | Epoch: {}".format(fold, epoch))

    completed_start_steps = epoch * len(emodb_train_loader)
    total_steps = EPOCHS * len(emodb_train_loader)


    for batch_idx, (feature, emotion, speaker) in enumerate(emodb_train_loader):

        # Assigned to DEVICE.
        features, emotion, speaker = feature.to(DEVICE),emotion.to(DEVICE), speaker.to(DEVICE)

        # Computing the training progress
        p = (batch_idx + completed_start_steps) / total_steps
        lambda_p = domain_adaptation_parameter(p)

        # Calculate speaker and emotion classification prediction
        conv_features = encoder(features)
        emotion_preds = emotion_classifier(conv_features)
        emotion_class_loss = cross_entropy_loss(emotion_preds, emotion)
        speaker_preds = speaker_classifier(conv_features)
        speaker_class_loss = cross_entropy_loss(speaker_preds, speaker)

        # Calculate total loss
        total_loss = emotion_class_loss - lambda_p * speaker_class_loss

        # Clear the gradient to prevent gradient accumulation
        encoder.zero_grad(set_to_none= True)
        emotion_classifier.zero_grad(set_to_none= True)
        speaker_classifier.zero_grad(set_to_none= True)

        # Computing the gradient
        total_loss.backward()

        # Update the weight
        emotion_optimizer.step()
        speaker_optimizer.step()
        encoder_optimizer.step()
```

Code Snippet 12 Main Section of Code responsible for Speaker Adversarial Training

After each epoch, the model will be evaluated. If the model shows an improvement with the validation dataset, the parameters (weights and biases) at that epoch will be saved and overwrite the previous saved parameters, otherwise the model will keep the previous saved parameters. At the end of 100 epochs, the model will then be tested using the best parameters saved. The entire process will then be repeated until 5-fold leave-two-speakers-out cross validation and testing is performed for Emo-DB experiments and 12-fold leave-two-speakers-out cross validation and testing is done for RAVDESS.

3.5 Comparison with network architecture without DAT

To study the effect of DAT, we must compare it to a network architecture not utilising DAT. Structurally, it has the same encoder and emotion classifier as our proposed network, but without the speaker classifier.

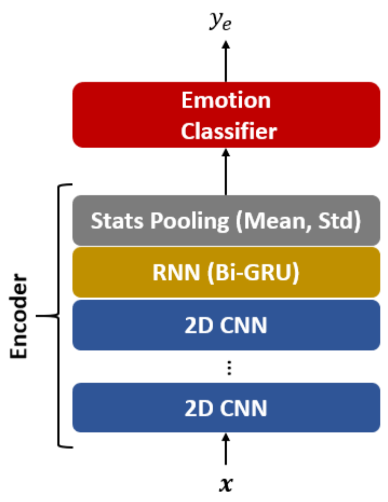
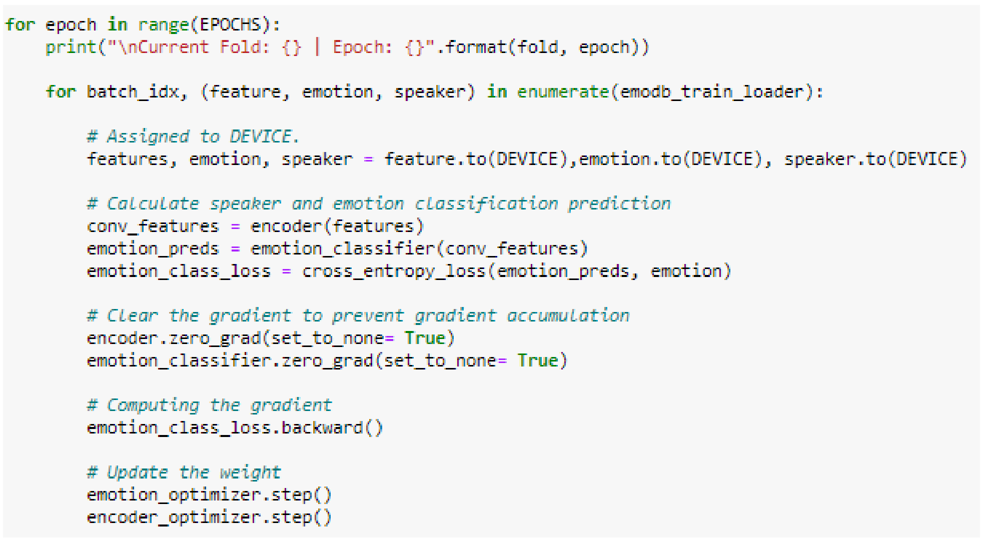


Figure 32 The proposed network architecture without DAT

There is also a difference in training approach. Since there is no speaker classifier, there will not be any gradient reversal. There will also be no need to compute the Cross Entropy Loss between the speaker prediction and speaker label as shown in the code snippet below.

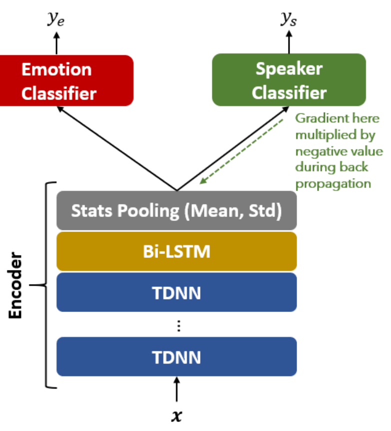


```
for epoch in range(EPOCHS):
    print("\nCurrent Fold: {} | Epoch: {}".format(fold, epoch))

    for batch_idx, (feature, emotion, speaker) in enumerate(emodb_train_loader):

        # Assigned to DEVICE.
        features, emotion, speaker = feature.to(DEVICE),emotion.to(DEVICE), speaker.to(DEVICE)

        # Calculate speaker and emotion classification prediction
        conv_features = encoder(features)
        emotion_preds = emotion_classifier(conv_features)
        emotion_class_loss = cross_entropy_loss(emotion_preds, emotion)

        # Clear the gradient to prevent gradient accumulation
        encoder.zero_grad(set_to_none= True)
        emotion_classifier.zero_grad(set_to_none= True)

        # Computing the gradient
        emotion_class_loss.backward()

        # Update the weight
        emotion_optimizer.step()
        encoder_optimizer.step()
```

Code Snippet 13 Main section of code without speaker adversarial training

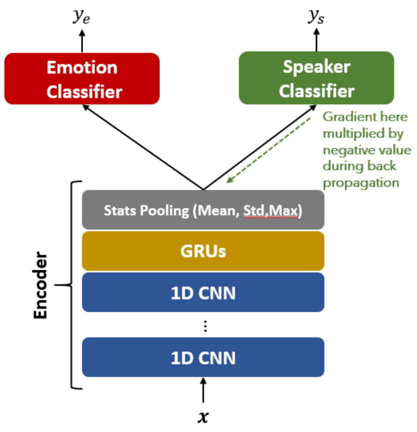
3.7 Comparison with state-of-the-art network architecture used in other speaker independent emotion recognition task

As mentioned earlier, DAT has also been investigated in removing speaker variability in speech by learning speaker invariant representations [17]. However, such network use 1D CNN with RNN. While the 1D CNN can capture the short-term temporal features before passing it to RNN which capture the long-term temporal features, they only capture one frequency component, which we feel is unsuitable for speaker independent speech recognition. This is because speech is composed of multiple frequencies and distinguishing those frequency component will help in distinguishing the emotions and speakers. With gradient reversal, the encoder will then help to make the encoder speaker independent. Hence, our encoder make use of 2D CNN followed by Bi-GRU which will capture both the multiple frequency components and temporal features.



y_e
y_s
Emotion Classifier
Speaker Classifier
Gradient here multiplied by negative value during back propagation
Stats Pooling (Mean, Std)
Bi-LSTM
TDNN
TDNN
Encoder
x

Figure 33 Model Architecture in [17]



y_e
y_s
Emotion Classifier
Speaker Classifier
Gradient here multiplied by negative value during back propagation
Stats Pooling (Mean, Std,Max)
GRUs
1D CNN
1D CNN
Encoder
x

Figure 34 Model Architecture in [18]

To understand how our 2D CNN biGRU encoder perform relative to them, comparisons were also done. To ensure fairness of comparison, we perform the same 5-fold-leave-two-speakers out cross validation and testing and 12-fold-leave-two-speakers out cross validation on Emo-DB and RAVDESS dataset respectively on their encoders. They are performed with the same type of input features mentioned in their respective papers. For [17], it will be MFCC and for [18], it will be logMFB, pitch and energy.

4. Experiment and Results

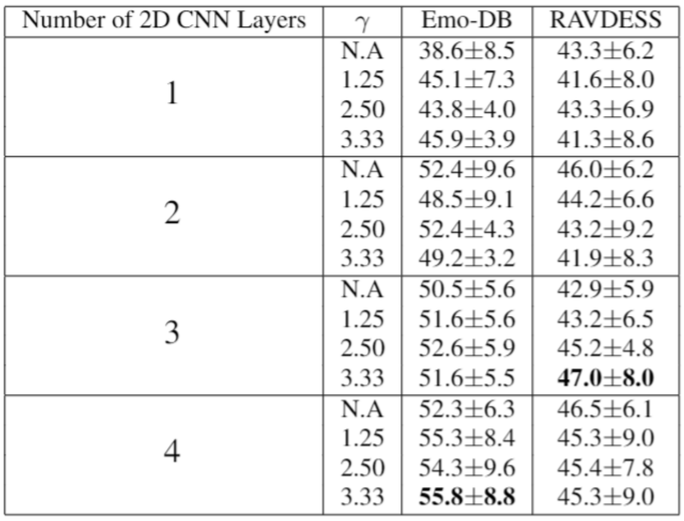
Table 1 shows the result of our 2D CNN Bi-GRU encoder with different number of CNN layers with MFCC features. For each depth, we compare the performance of our 2D CNN Bi-GRU with different value of γ, where a higher γ value will result in a higher rate and degree of speaker adversarial training. Comparisons were also made to our 2D CNN biGRU encoder without DAT. Since the 2D CNN Bi-GRU encoder without DAT does not have a speaker classifier connected to it, there will be no gradient reversal occurring, and a γ value is not applicable to it.

From Table 1, it is quite apparent that having lesser number of CNN in our 2D CNN Bi-GRU encoders yield better result. Our encoder trained with DAT also seems to perform better than without DAT. However, the depth of the encoder was also critical. We note that the benefits of training our 2D CNN Bi-GRU encoders with DAT diminish as the number of CNN layers increase. This is especially true for the case with the RAVDESS dataset. When the number of 2D CNN layers is 4, the 2D CNN Bi-GRU trained with DAT does not seem offer much additional performance compared to without DAT for RAVDESS. We found that one CNN layer depth is optimal for both datasets for DAT for SIER.

Number of 2D CNN Layers	γ	Emo-DB	RAVDESS
1	N.A	67.3±6.5	56.3±8.1
	1.25	72.1±8.4	60.6±10.3
	2.50	70.6±5.9	**61.2±8.4**
	3.33	**73.6±7.1**	60.6±10.6
2	N.A	68.3±9.6	56.3±9.8
	1.25	68.0±8.0	59.4±7.6
	2.50	71.0±7.6	60.9±8.5
	3.33	67.6±7.4	60.5±9.9
3	N.A	65.7±10.6	55.8±8.8
	1.25	65.1±7.2	53.7±6.7
	2.50	66.0±5.5	59.4±7.6
	3.33	66.6±5.6	58.9±8.5
4	N.A	59.6±5.3	54.0±11.3
	1.25	66.0±6.9	54.2±8.7
	2.50	62.9±9.6	54.1±9.3
	3.33	65.2±7.2	51.3±11.1

Table 1 Leave 2 speaker out CV accuracy of Emo-DB and RAVDESS for MFCC features

Table 2 shows the result of our 2D CNN Bi-GRU encoder with different number of CNN layers with LMS features.



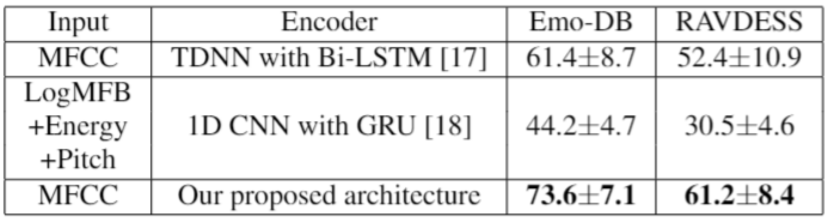
Number of 2D CNN Layers	γ	Emo-DB	RAVDESS
1	N.A	38.6±8.5	43.3±6.2
	1.25	45.1±7.3	41.6±8.0
	2.50	43.8±4.0	43.3±6.9
	3.33	45.9±3.9	41.3±8.6
2	N.A	52.4±9.6	46.0±6.2
	1.25	48.5±9.1	44.2±6.6
	2.50	52.4±4.3	43.2±9.2
	3.33	49.2±3.2	41.9±8.3
3	N.A	50.5±5.6	42.9±5.9
	1.25	51.6±5.6	43.2±6.5
	2.50	52.6±5.9	45.2±4.8
	3.33	51.6±5.5	**47.0±8.0**
4	N.A	52.3±6.3	46.5±6.1
	1.25	55.3±8.4	45.3±9.0
	2.50	54.3±9.6	45.4±7.8
	3.33	**55.8±8.8**	45.3±9.0

Table 2 Leave 2 speaker out CV accuracy of Emo-DB and RAVDESS for LMS features

When it comes to LMS, Table2 shows that greater numbers of CNN in our 2D CNN Bi-GRU encoders yield better result. For the Emo-DB experiments, apart from the case where layers = 2, it is quite apparent that our encoder with DAT performs significantly better than without DAT. However, these γ values do not seem to be suitable for the RAVDESS dataset as most of the time, training without DAT yield better results.

Comparing Table 1 to Table 2, it is apparent that the same architecture does not seem to be suitable for LMS features as the performance is significantly much more inferior compared to that with MFCC features.

Table 3 shows the comparison of our proposed encoder with other encoders using DAT for Speaker Invariant Emotion Recognition. It shows that our encoder achieves the best performance compared to their encoders in both RAVDESS and Emo-DB dataset, confirming the superiority of encoders utilising 2D filters for SIER tasks.

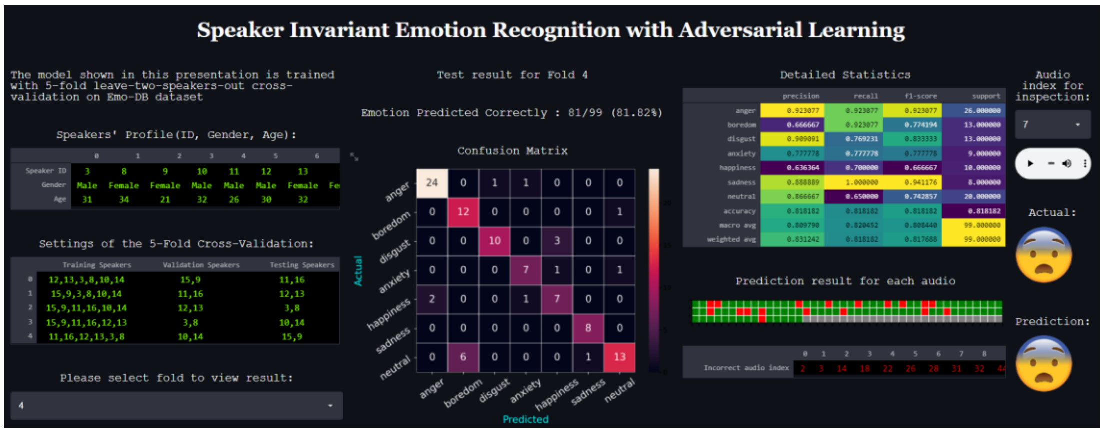


Input	Encoder	Emo-DB	RAVDESS
MFCC	TDNN with Bi-LSTM [17]	61.4±8.7	52.4±10.9
LogMFB +Energy +Pitch	1D CNN with GRU [18]	44.2±4.7	30.5±4.6
MFCC	Our proposed architecture	**73.6±7.1**	**61.2±8.4**

Table 3 Comparison of our proposed encoder with other encoders using DAT for SIER

5. Web Application for Demonstration

For Demonstration purposes, we have created a simple web application with Streamlit, an open-source app framework for Machine Learning and Data Science teams.



Speaker Invariant Emotion Recognition with Adversarial Learning

The model shown in this presentation is trained with 5-fold leave-two-speakers-out cross-validation on Emo-DB dataset

Speakers' Profile(ID, Gender, Age):

	0	1	2	3	4	5	6	
Speaker ID	3	8	9	10	11	12	13	
Gender	Male	Female	Female	Male	Male	Male	Female	Fe
Age	31	34	21	32	26	30	32	

Settings of the 5-Fold Cross-Validation:

	Training Speakers	Validation Speakers	Testing Speakers
0	12,13,3,8,10,14	15,9	11,16
1	15,9,3,8,10,14	11,16	12,13
2	15,9,11,16,10,14	12,13	3,8
3	15,9,11,16,12,13	3,8	10,14
4	11,16,12,13,3,8	10,14	15,9

Please select fold to view result:

4

Test result for Fold 4

Emotion Predicted Correctly : 81/99 (81.82%)

Confusion Matrix

Actual \ Predicted	anger	boredom	disgust	anxiety	happiness	sadness	neutral
anger	24	0	1	1	0	0	0
boredom	0	12	0	0	0	0	1
disgust	0	0	10	0	3	0	0
anxiety	0	0	0	7	1	0	1
happiness	2	0	0	1	7	0	0
sadness	0	0	0	0	0	8	0
neutral	0	6	0	0	0	1	13

Detailed Statistics

	precision	recall	f1-score	support
anger	0.923077	0.923077	0.923077	26.000000
boredom	0.666667	0.923077	0.774194	13.000000
disgust	0.909091	0.769231	0.833333	13.000000
anxiety	0.777778	0.777778	0.777778	9.000000
happiness	0.636364	0.700000	0.666667	10.000000
sadness	0.888889	1.000000	0.941176	8.000000
neutral	0.866667	0.650000	0.742857	20.000000
accuracy	0.818182	0.818182	0.818182	0.818182
macro avg	0.809790	0.820452	0.808440	99.000000
weighted avg	0.831242	0.818182	0.817688	99.000000

Prediction result for each audio

	0	1	2	3	4	5	6	7	8	
Incorrect audio index	2	3	14	18	22	26	28	31	32	44

Audio index for inspection:

7

Actual:

Prediction:

The left side of this application shows the speakers 'profile (ID, Gender and Age). It also shows the settings of the leave-two-speakers cross-validation. We can select the fold to view the result of each fold.

After selecting the fold, the application will generate the test result for that fold. We can see the confusion matrix and detailed statistics such as precision, recall, F1-score and support for each emotion in the dataset.

Under prediction result for each audio, we can see which audio (in terms of index starting from 0) our model predicted correctly and wrongly. The green grid means that the audio is predicted correctly, while the red grid means that the audio is predicted wrongly. There is even a table that highlights which audio is predicted incorrectly.

We can also listen to the audio in this fold. One use case for this is to listen to the audio that our model predicted incorrectly. We can then listen to the audio ourselves to judge if that audio is hard to predict even by human judgement. The application will also shows the actual emotion label and the predicted emotion of that audio by our model.

6. Conclusion and Future Work

Domain adversarial training (DAT) can be used to learn speaker invariant representations in Speaker Independent Emotion Recognition (SIER) tasks. In this project, we have demonstrated that our proposed 2D Convolutional Neural Network (CNN) with Bi-directional Gated Recurrent Unit (Bi-GRU) architecture outperforms 1D TDNN with Bi-directional Long Short-Term Memory (Bi-LSTM) architecture[17] and 1D CNN with GRU architecture[18] and achieve state-of-the-art accuracy for SIER. This shows that capturing the multiple frequency components and temporal features with 2D filters are important in speaker invariant emotion recognition.

Throughout our experiments, we came to the realisation that the input features matter a lot in SIER. For example, we have realised that in our experiment, MFCC work better than LMS and LogMFB with energy and pitch. In addition, we also realised that the dimension of the filter matter. Hence, other than the model architecture, it is prudent to have the future work investigate the feature engineering of input features as well. One way of doing such feature engineering is to simply try out different frame size, hop size and window function in creating the MFCC features before feeding into the encoder. This is not so different from hyperparameter selection in deep neural network architecture. Going further, one direction is to encode the dynamics of the power spectrum, for example, the trajectories of MFCCs over time with the delta (differential) and delta-delta (acceleration) coefficients into the input features. With that, we can investigate the use of higher dimensional filter, i.e., 3D or 4D filters in the encoder to capture such information for SIER task.

7. References

[1] D. Yu, K. Han, and I. Tashev, “Speech Emotion Recognition Using Deep Neural Network and Extreme Learning Machine Spatial sound View project Speech Emotion Recognition Using Deep Neural Network and Extreme Learning Machine,” 2014. [Online]. Available: https://www.researchgate.net/publication/267213794.

[2] T. Seehapoch and S. Wongthanavasu, “Speech emotion recognition using Support Vector Machines,” Jan. 2013, doi: 10.1109/KST.2013.6512793.

[3] Y.-Y. Kao *et al.*, “Automatic Detection of Speech Under Cold Using Discriminative Autoencoders and Strength Modeling with Multiple Sub-Dictionary Generation,” Sep. 2018, doi: 10.1109/IWAENC.2018.8521319.

[4] L. Chao, J. Tao, M. Yang, and Y. Li, “Improving generation performance of speech emotion recognition by denoising autoencoders,” Sep. 2014, doi: 10.1109/ISCSLP.2014.6936627.

[5] A. Batliner *et al.*, “The INTERSPEECH 2010 paralinguistic challenge The INTERSPEECH 2010 Paralinguistic Challenge *,” 2010. [Online]. Available: https://www.researchgate.net/publication/221481381.

[6] G. Trigeorgis *et al.*, “Adieu features? End-to-end speech emotion recognition using a deep convolutional recurrent network,” Mar. 2016, doi: 10.1109/ICASSP.2016.7472669.

[7] S. Mirsamadi, E. Barsoum, and C. Zhang, “Automatic speech emotion recognition using recurrent neural networks with local attention,” Mar. 2017, doi: 10.1109/ICASSP.2017.7952552.

[8] J. Zhao, X. Mao, and L. Chen, “Speech emotion recognition using deep 1D &amp; 2D CNN LSTM networks,” *Biomedical Signal Processing and Control*, vol. 47, Jan. 2019, doi: 10.1016/j.bspc.2018.08.035.

[9] M. Chen, X. He, J. Yang, and H. Zhang, “3-D Convolutional Recurrent Neural Networks With Attention Model for Speech Emotion Recognition,” *IEEE Signal Processing Letters*, vol. 25, no. 10, Oct. 2018, doi: 10.1109/LSP.2018.2860246.

[10] H. Meng, T. Yan, F. Yuan, and H. Wei, “Speech Emotion Recognition From 3D Log-Mel Spectrograms With Deep Learning Network,” *IEEE Access*, vol. 7, 2019, doi: 10.1109/ACCESS.2019.2938007.

[11] J. Rybka and A. Janicki, “Comparison of speaker dependent and speaker independent emotion recognition,” *International Journal of Applied Mathematics and Computer Science*, vol. 23, no. 4, Dec. 2013, doi: 10.2478/amcs-2013-0060.

[12] I. J. Goodfellow *et al.*, “Generative Adversarial Networks,” Jun. 2014, [Online]. Available: http://arxiv.org/abs/1406.2661.

[13] S. Sahu, R. Gupta, G. Sivaraman, W. AbdAlmageed, and C. Espy-Wilson, “Adversarial Auto-Encoders for Speech Based Emotion Recognition,” Aug. 2017, doi: 10.21437/Interspeech.2017-1421.

[14] S. Latif, M. Asim, R. Rana, S. Khalifa, R. Jurdak, and B. W. Schuller, “Augmenting Generative Adversarial Networks for Speech Emotion Recognition,” Oct. 2020, doi: 10.21437/Interspeech.2020-3194.

[15] S. E. Eskimez, D. Dimitriadis, R. Gmyr, and K. Kumanati, “GAN-based data generation for speech emotion recognition,” in *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2020, vol. 2020-October, pp. 3446–3450, doi: 10.21437/Interspeech.2020-2898.

[16] M. Abdelwahab and C. Busso, “Domain Adversarial for Acoustic Emotion Recognition,” Apr. 2018, [Online]. Available: http://arxiv.org/abs/1804.07690.

[17] M. Tu, Y. Tang, J. Huang, X. He, and B. Zhou, “Towards adversarial learning of speaker-invariant representation for speech emotion recognition,” Mar. 2019, [Online]. Available: http://arxiv.org/abs/1903.09606.

[18] H. Li, M. Tu, J. Huang, S. Narayanan, and P. Georgiou, “Speaker-invariant Affective Representation Learning via Adversarial Training,” Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.01533.

[19] F. Burkhardt, A. Paeschke, M. Rolfes, W. Sendlmeier, and B. Weiss, “A Database of German Emotional Speech.” [Online]. Available: http://www.expressive-speech.net/emodb/.

[20] S. R. Livingstone and F. A. Russo, “The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English,” *PLOS ONE*, vol. 13, no. 5, May 2018, doi: 10.1371/journal.pone.0196391.

[21] Y. Ganin *et al.*, “Domain-Adversarial Training of Neural Networks,” May 2015, [Online]. Available: http://arxiv.org/abs/1505.07818.

[22] N. Upadhyay and A. Karmakar, “Speech Enhancement using Spectral Subtraction-type Algorithms: A Comparison and Simulation Study,” *Procedia Computer Science*, vol. 54, 2015, doi: 10.1016/j.procs.2015.06.066.

8. Appendix

8.1 Paper submitted to INTERSPEECH 2021

Speaker-Invariant Emotion Recognition with Adversarial Learning

Bryan Leow Xuan Zhen, L. L. Chamara Kasun, Ahn Chung Soo, Jagath C. Rajapakse

School of Computer Science and Engineering, Nanyang Technological University, Singapore

{xleow002, chamarakasun, csahn, asjagath}@ntu.edu.sg

Abstract

Performances of current methods for speech emotion recognition are dependent on whether the speakers are present in the training dataset. In this paper, we propose adversarial learning (AL) network for speaker-invariant emotion recognition (SIER) task. SIER is achieved in a network consisting of an encoder, an emotion classifier, and a speaker classifier, and implementing an adversarial learning strategy to learn representations that are invariant for speaker characteristics. We argue that the representation realized by AL network are independent of the speakers and therefore are conducive for SIER. Earlier, adversarial learning has been used for SIER and we present an improved encoder that demonstrated state-of-the-art performances for SIER on Emo-DB and RAVDESS datasets.

Index Terms: convolutional neural networks, domain adversarial training, recurrent neural networks, speaker independent emotion recognition.

1. Introduction

Speech emotion recognition (SER) aims at recognition of speakers' emotion from speech signals. SER algorithms typically begin with the extraction of features from speech and are then followed by classification of speech signals into emotions. Features extracted include high-level features such as those specificed by Interspeech Para-linguistics Challenge 2010 [1] and low-level features such as spectrogram features. Recently, deep learning architectures have become popular for SER and shown that they can directly learn from low-level features without resorting to time-consuming processes of creating high-level features. Direct use of Mel-scale spectrograms have shown state-of-the-art accuracy in speech recognition task with deep learning [2].

Performances of typical SER methods are dependent on whether the speaker is present or absent in the training set. Ideally, SER algorithms are to perform irrespective of whether speaker is new to the algorithm. In this paper, we focus on Speaker-Invariant Emotion Recognition (SIER) where the training and testing data come from different speakers. The aim of SIER is to create a model for speech data spoken from a set of speakers and apply the model to predict emotions from speech from a different set of speakers. SIER is more challenging than SER as SIER algorithms must learn a speaker invariant representation where speakers' identity is removed.

Typically for SER tasks, the training data is collected on an environment (source domain) different from the environment (target domain) where the predictions are performed. If source and target domain distributions are different, SER algorithms perform poorly during testing. Ideally, SER methods need to learn representations that are common to either domain. In order to address the issue of variability between training and testing data, adversarial learning approaches, inspired by Generative Adversarial Network (GAN) [3], have been proposed for SER. One approach is to use GAN to generate synthetic data to augment the training set [4, 5, 6]. The models trained with the augmented training data have shown to perform better than models trained with only original data. Alternatively, domain adversarial training can be used to resolve the disparity between training and testing environments or domains [7]. Domain adversarial networks using AL was trained with source data and to predict on target data which is from another source different from the source data [8, 9]. Adversarial has also been investigated in removing speaker variability in speech by learning speaker invariant representations [10, 11].

In this paper, we propose an adversarial learning (AL) network that learns representations independent of speaker characteristics for SIER task. The proposed AL network consists of an encoder that learns speaker-invariant representations, an emotion classifier to predict emotion labels, and a speaker classifier that helps remove speaker variability. Our work extends the work initiated by [10, 11] by proposing a novel decoder to learn speaker-invariant representations. By minimizing the accuracy of speaker classifier while maximizing the accuracy of emotion prediction, the proposed AL network is agnostic to speaker characteristics. Our encoder uses 2-dimensional (2D) Convolutional Neural Network (CNN) layers followed by bidirectional Gated Recurrent Units (biGRU). We also perform experiments to determine the required depth of the proposed 2D CNN biGRU encoder.

The popularity deep learning approaches to SER can be attributed to their usage of the Mel-scale to compute features such as MFCC where equal distance in pitch sound equally distant to the listener. This is important because humans do not perceive frequency on a linear scale and can distinguish low frequencies at a better resolution than high frequencies, i.e., a human can tell apart a 500Hz and 1000Hz signal but cannot tell the difference between a 1Khz and 1.5kHz signal. Hence, MFCC aim to capture this non-linear perception of sound with the frequencies spaced according to perception. MFCC represents speech as a set of frequencies that vary along with time. The number of frequencies captured can be determined by the user. However, previous adversarial learning for SIER used a 1-D neural architecture that had learned short-term temporal features capturing only statistical features such as energies of frequency components [11]. In order to incorporate multiple frequency components for SIER, we utilize 2D sprectral features given by MFCC coefficients gathered over a time interval.

Deep learning architectures consisting of Deep Neural Networks (DNN), CNN, and Recurrent Neural Networks (RNN) have been used for SER. A typical speech encoder consists of several DNN or CNN layers followed by RNN layers made up of Long Short-Term Memory (LSTM) units or GRUs. Inspired by the success of 2D CNN LSTM for SER [12], we propose 2D CNN biGRU architecture as the encoder for AL network. Further, we empirically determine the required depth for SIER. The CNN captures local contextual information from spectral features and biGRU captures the temporal dependencies. Previously, 1D CNN [11] and Time-delayed Neural Networks (TDNN) [10] have been used for SIER, which did not capture the necessary contextual information from speech. We demonstrate that 2D CNN biGRU encoder achieves state-of-the-art performances on two benchmark datasets for SIER.

2. Related work

SIER tasks typically have a limited number of training data due to the difficulty of collecting and labeling speech emotion data. Hence, GAN [3] approaches have been introduced to SIER tasks to generate data for augmentation. Traditional GAN architectures use a random Gaussian vector as a seed to generate data, assuming a Gaussian generative model for speech. However, due to the high variability of speech emotion data, speech emotion data cannot be modelled as coming from a simple Gaussian distribution [5]. Hence, a linearly mixed speech emotion data are used to generate data in GAN for SER. Another approach of generating speech emotion data is to use an autoencoder to generate latent vector [6]. And use this latent vector added with random Gaussian noise as a seed to a GAN to generate speech emotion data. This approach has been shown of capable of generating speech emotion belonging to a specific category such as anger or happy.

Adversarial learning has also been attempted to resolve the issue of variability of data for SER [7]. Speech emotion data can be collected from different domains such as with German and English speakers. Machine learning algorithms trained on German speech emotion data have difficulty in predicting emotion of English speech emotion data. Domain adversarial training has been applied to remove the variation of data, caused due to the variability across different domains [8]. This is achieved by using a encoder, emotion classifier and domain classifier with gradient reversal. Domain classifier aims to remove the variability of the domain. This same concept of learning a domain invariant representation has been extended to learning a speaker invariant representation by changing the domain classifier to speaker classifier with gradient reversal [10, 11].

3. Methodology

The proposed AL network architecture is illustrated in figure 1, which consists of three components: (i) an encoder made out of 2D CNN and biGRU layers, which generates speaker invariant representations; (ii) an emotion classifier that predicts emotional labels; and (iii) a speaker classifier with gradient reversal, which removes speaker variability.

3.1. Encoder

The network takes MFCC features of speech in a time window size of T . Input speech emotion data $x = (x(t))$ where $x(t)$ denotes the features extracted at time t, $x \in \mathbb{R}^{T\times n}$ and n denotes the number of features. The encoder consist of a one or more 2D CNN layers followed by one biGRU layer. The 2D CNN aims to learn short-term frequency features as well as temporal information as speech contain multiple frequencies. 2D CNN consist of (i) a 2D convolutional layer; (ii) a batch normalization layer; and (iii) a max pooling layer.

Let w_k^1 denote the filter weight connected to k the feature map at the convolutional layer. The output h_k^1 of the k the feature is given by

$$h_k^1 = x \circledast w_k^1 \tag{1}$$

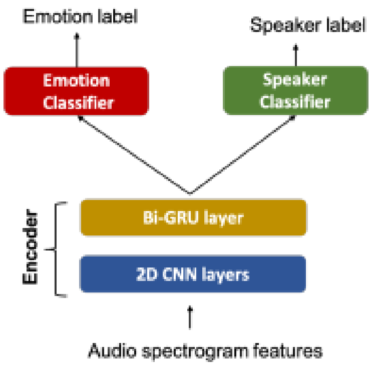


Figure 1: *Proposed adversarial learning (AL) network architecture. The encoder consists of 2D CNN layers and a bidirectional GRU (biGRU) layer. Emotion and speaker classifier consist of two hidden layer fully connected neural network layers followed by a softmax layer.*

where $\circledast$ denote the convolution operation. The convolution layer output is processed by a batch-normalization layer, which is processed by dropout and exponential linear units (eLU). The batch-normalization layer output h_k^2 for k feature map is given by

$$h_k^2 = \mathtt{eLU}\left(\mathtt{dropout}\left(w_k^2 \cdot \mathtt{BN}(h_k^1) + b_k^2\right)\right) \tag{2}$$

where $\cdot$ denotes the element-wise multiplication, BN denotes batch-normalization function that normalizes the data by subtracting the mean and dividing by the standard deviation over a batch. w_k^2 and b_k^2 denote learnable parameters. `dropouts` and `eLU` denote dropout operation and eLU activation function. Max-pooling layer output h_k^2 is given by

$$h_k^3 = \mathtt{pool}\left(h_k^2\right) \tag{3}$$

Let $h_k^3 \in \mathbb{R}^{T\times n^3}$ where n_3 is the 2D CNN output features.

The Bi-directional Gated Recurrent Unit (biGRU) learns the long-term temporal relationship of the short-term frequency and temporal features and in this work we used a biGRU as the recurrent neural network. To achieve this, the convolution layer output $h^3 = (h_k^3) \in \mathbb{R}^{K\times T_s\times n^3}$ is reshaped as $h^3(t) \in \mathbb{R}^{K\times n^3}$. The RNN output $h^4(t)$ is given by:

$$h^4(t) = \mathtt{biGRU}(h^3(t), h^4(t-1)) \tag{4}$$

where $h^4(t) \in \mathbb{R}^{n^4}$ and n^4 is the number of hidden neurons in bi-GRU. The output of the stats pooling layer $h^5 \in \mathbb{R}^{2*n^4}$ is given by:

$$h^5 = \mathtt{stats\_pool}(h^4) \tag{5}$$

where `stats_pool` function calculates the mean and standard deviation of h^4 along the time dimension and concatenates along the feature dimension.

3.2. Emotion and Speaker Classifiers

Emotion and speaker classifiers consist of two fully connected layers and each layer performs batch-normalization and dropout, and is processed with ReLU activation function. Output layer is a softmax layer.

Emotion classifier predicts the emotion labels y_e from the stats pooling layer output h^5 as:

$$\begin{aligned} h^6 &= \texttt{ReLU}\left(\texttt{dropout}\left(W^6\texttt{BN}\left(V^6h^5 + c^6\right) + b^6\right)\right) \\ h^7 &= \texttt{ReLU}\left(\texttt{dropout}\left(W^7\texttt{BN}\left(V^7h^6 + c^7\right) + b^7\right)\right) \\ y_e &= \texttt{softmax}\left(W^8h^7 + b^8\right) \end{aligned} \tag{6}$$

Speaker classifier predicts speaker labels y_s from the stats pooling layer output h^5 as:

$$\begin{aligned} h^9 &= \texttt{relu}\left(\texttt{dropout}\left(W^9\texttt{BN}\left(V^9h^5 + c^9\right) + b^9\right)\right) \\ h^{10} &= \texttt{relu}\left(\texttt{dropout}\left(W^{10}\texttt{BN}\left(V^{10}h^9 + c^{10}\right) + b^{10}\right)\right) \\ y_s &= \texttt{softmax}\left(W^{11}h^{10} + b^{11}\right) \end{aligned} \tag{7}$$

3.3. Adversarial Learning

We minimize the cross entropy loss of the emotion classifier as:

$$J_e = -E_x[d_e log(y_e)] \tag{8}$$

where d_e is the emotion labels and E_x is the expectation over data x. The cross-entropy loss J_s of speaker classifier is given by

$$J_s = -E_x[d_s log(y_s)] \tag{9}$$

where d_s are the speaker identity labels.

During training, the parameters of the underlying deep feature mapping are optimised to minimise the loss of the emotion classifier J_e and to maximise the loss of the speaker classifier J_s. The latter is enabled with the use of the gradient reversal that leaves the features unchanged during forward propagation and reverses the gradient by multiplying it by a negative scalar λ during the backpropagation, thus working in an adversarial manner to the domain classifier and encourages the emergence of speaker-invariant features.

The overall loss that is minimized during learning is given by

$$J = J_e - \lambda J_s \tag{10}$$

where $\lambda = \frac{2}{1+exp(-\gamma p)} - 1$, γ is a positive annealing hyper parameter, and p is the percentage of training. Importance factor of the speaker classifier λ gradually increases from 0 to at most 1 with the training progress since the negative gradient can hamper the initial weight updates [7]. The rate and degree of speaker adversarial training occurs can be adjusted by changing γ, where a higher γ value will result in a higher rate and degree of speaker adversarial training.

4. Experiments and Results

In this section, we investigate the efficacy of the AL network encoder on two benchmark datasets for SIER: Emo-DB [13] and RAVDESS datasets [14]. The performances of the our network is compared with two existing methods that uses AL for SIER: (i)1D TDNN biLSTM [10] ; (ii) 1D CNN GRU [11].

All the experiments were carried out on a DGX server with two Xeon E5-2698 v4 clocked at 2.2 GHz, 512 GB RAM and eight Nvidia V100 32 GB graphic cards running Ubuntu 16.04. The scripts were written in Python using Pytorch package.

4.1. Datasets and feature extraction

Emo-DB dataset contain speech recording of 7 emotions representing anger, boredom, disgust, fear, happy, sad and neutral, from 10 German speaking persons. There were total of 535 speech samples. For SIER, we performed 5-fold leave-two-speakers-out for cross-validation and testing. That is, in each fold, 6 speakers were selected for training, 2 speakers for validation and 2 speakers for testing. The ratio of male to female speakers in training, validation, and testing was set to 1:1.

Table 1: *Leave-two-speaker-out cross-validation accuracy (%) of adversarial learning network with 2D CNN biGRU encoder for speaker-invariant emotion recognition on Emo-DB and RAVDESS datasets at different number of CNN layers and annealing parameter γ values.*

No. of CNN layers	γ	Emo-DB	RAVDESS
1	without AL	67.3±6.5	56.3±8.1
	1.25	72.1±8.4	60.6±10.3
	2.50	70.6±5.9	**61.2±8.4**
	3.33	**73.6±7.1**	60.6±10.6
2	without AL	68.3±9.6	56.3±9.8
	1.25	68.0±8.0	59.4±7.6
	2.50	71.0±7.6	60.9±8.5
	3.33	67.6±7.4	60.5±9.9
3	without AL	65.7±10.6	55.8±8.8
	1.25	65.1±7.2	53.7±6.7
	2.50	66.0±5.5	59.4±7.6
	3.33	66.6±5.6	58.9±8.5
4	without AL	59.6±5.3	54.0±11.3
	1.25	66.0±6.9	54.2±8.7
	2.50	62.9±9.6	54.1±9.3
	3.33	65.2±7.2	51.3±11.1

RAVDESS dataset contains speech recordings of 8 emotions representing anger, calm, surprised, fear, happy, sad, neutral and disgust, from 24 professional actors with north American English accent. There were total of 1440 speech samples. We performed 12-fold leave-two-speakers-out cross validation and testing by selecting 20 speakers for training, 2 for validation and 2 for testing. The ratio of male to female speakers in training, validation and testing was set to 1:1.

Speech recordings in both datasets were trimmed and filtered to remove silence and background noise. The recordings in each dataset were then padded with zeros to ensure that they are of the same length as the longest audio recording in the dataset. The pre-processed recordings were then converted to MFCC features with the following settings: frame size of 2048 frames, hop size of 512 frames, and Hann Window of 2048 frames as a windowing function to minimise spectral leakage. We only retained the first 20 MFCC coefficients.

4.2. Parameter initialization

Hyper-parameter selection for the neural network architecture was performed using the grid search method and the parameters that yielded the best accuracy were selected. We chose γ from $[1.25, 2.5, 3.33]$ and the number of layers from 1 to 4 in 2D CNN with biGRU encoder. We used convolutional filter size from $[5 \times 5, 2 \times 2]$, number of convolutional filters $[128, 64]$, number of hidden neurons in GRU $[256, 128]$ and the number of hidden neurons in the fully connected layer $[128, 64]$. We set the convolutional stride two 2 and the max pooling size 2×2 and stride pooling stride of 2.

Table 2: *Comparison of leave-two-speaker-out cross-validation accuracy (%) of different encoders for AL for speaker-invariant emotion recognition.*

Input features	Encoder	Emo-DB	RAVDESS
MFCC	TDNN biLSTM [10]	61.4±8.7	52.4±10.9
LogMFB, energy, pitch	1D CNN GRU [11]	44.2±4.7	30.5±4.6
MFCC	2D CNN biGRU	**73.6±7.1**	**61.2±8.4**

4.3. Adversarial learning with 2D CNN biGRU encoder

Table 1 shows the result of our 2D CNN biGRU encoder with different number of CNN layers. For each depth, we compare the performance of our 2D CNN biGRU with different value of γ, where a higher γ value resulted in a higher rate and degree of speaker adversarial training. Comparison were also made to our 2D CNN biGRU encoder without AL since the 2D CNN biGRU encoder without AL did not have a speaker classifier connected to it.

As seen Table 1, it is quite apparent that having lesser number of CNN in our 2D CNN biGRU encoders yield better result. Our encoder trained with AL significantly outperformed training without AL. However, the depth of the encoder was also critical. We note that the benefits of training our 2D CNN biGRU encoders with AL diminish as the number of CNN layers increase. This is especially true for the case with the RAVDESS dataset. When the number of 2D CNN layers is 4, the 2D CNN biGRU trained with AL does not seem offer much additional performance compared to without AL for RAVDESS. We found that one CNN layer depth is optimal for both datasets for AL for SIER.

4.4. Comparison with existing AL architectures

To understand how our 2D CNN biGRU encoder perform relative to other encoders utilising AL in SIER, comparisons to recent methods using AL [10, 11] were performed. To ensure fairness of comparison, we performed the same 5-fold leave-two-speakers-out cross-validation and testing and 12-fold-leave-two-speakers-out cross-validation and testing on Emo-DB and RAVDESS dataset, respectively, on the respective encoders. Table 2 shows that our proposed architecture achieves the best performance compared to existing encoders in both RAVDESS and Emo-DB dataset, confirming the superiority of encoders utilising 2D convolution filters for SIER tasks.

5. Conclusions

We presented an AL network to learn speaker-invariant representations for SER. The network learns representation maximizing emotion classification and minimizing the sensitivity for speaker characteristics. We demonstrated that proposed 2D CNN biGRU encoder outperformed existing 1D TDNN biLSTM and 1D CNN GRU and achieved state-of-the-art accuracy for SIER. Our AL network is useful when SER is to be trained on limited amount of speakers and be tested on unseen speakers.

6. Acknowledgements

We would like to thank Singtel Cognitive and Artificial Intelligence Lab for Enterprises Nanyang technological University Singapore for providing access to the GPU server.

7. References

[1] B. Schuller, S. Steidl, A. Batliner, F. Burkhardt, L. Devillers, C. Muller, and S. S. Narayanan, "The interspeech 2010 paralinguistic challenge," in *Proceedings of INTERSPEECH*, sep 2010.

[2] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech - Scaling up end-to-end speech recognition." *CoRR abs/1602.00985*, vol. cs.CL, 2014.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014.

[4] S. Sahu, R. Gupta, G. Sivaraman, C. Espy-Wilson, and W. AbdAlmageed, "Adversarial Auto-Encoders For Speech Based Emotion Recognition," in *Proceedings of INTERSPEECH*, 2017.

[5] S. Latif, M. Asim, R. Rana, S. Khalifa, R. Jurdak, and B. W. Schuller, "Augmenting Generative Adversarial Networks for Speech Emotion Recognition," in *Proceedings INTERSPEECH*, 2020, pp. 521–525.

[6] S. E. Eskimez, D. Dimitriadis, R. Gmyr, and K. Kumanati, "GAN-Based Data Generation for Speech Emotion Recognition," in *Proceedings INTERSPEECH*, 2020, pp. 3446–3450.

[7] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-Adversarial Training of Neural Networks," *Journal of Machine Learning Research*, vol. 17, no. 1, p. 2096–2030, Jan 2016.

[8] M. Abdelwahab and C. Busso, "Domain Adversarial for Acoustic Emotion Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, 04 2018.

[9] J. Parry, D. Palaz, G. Clarke, P. Lecomte, R. Mead, M. Berger, and G. Hofer, "Analysis of Deep Learning Architectures for Cross-Corpus Speech Emotion Recognition," in *Proceedings of INTERSPEECH*, 2019, pp. 1656–1660.

[10] M. Tu, Y. Tang, J. Huang, X. He, and B. Zhou, "Towards adversarial learning of speaker-invariant representation for speech emotion recognition," 2019.

[11] H. Li, M. Tu, J. Huang, S. Narayanan, and P. Georgiou, "Speaker-Invariant Affective Representation Learning via Adversarial Training," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020, pp. 7144–7148.

[12] J. Zhao, X. Mao, and L. Chen, "Speech emotion recognition using deep 1d & 2d cnn lstm networks," *Biomedical Signal Processing and Control*, vol. 47, pp. 312 – 323, 2019.

[13] F. Burkhardt, A. Paeschke, M. A. Rolfes, W. F. Sendlmeier, and B. Weiss, "A database of German emotional speech," in *Proceedings of INTERSPEECH*, 2005, pp. 1517–1520.

[14] S. R. Livingstone and F. A. Russo, "The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English," *PLOS ONE*, vol. 13, no. 5, pp. 1–35, 05 2018.