DEVELOPING A SPEECH EMOTION RECOGNITION SOLUTION USING ENSEMBLE LEARNING FOR CHILDREN WITH AUTISM SPECTRUM DISORDER TO HELP IDENTIFY HUMAN EMOTIONS

by

Rezwan Matin, B.S.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Engineering
December 2020

DEDICATION

To my father, who inspired me to become an engineer and make this world a better place.

ACKNOWLEDGEMENTS

The work accomplished in this thesis would not have been possible without the help of my mentors. At first, I would like to thank my research advisor Dr. Damian Valles, who introduced me to machine learning and allowed me to pursue this area of research. Without his support, I would not have been able to overcome the hurdles I have faced throughout my work. Next, I would like to thank Dr. Vishu Viswanathan, my Co-advisor. Because of Dr. Viswanathan's guidance I was able to learn a lot of speech signal processing concepts within a short period of time. I would also like to thank Dr. Maria Resendiz, whose passion for this research motivated me to give my best effort. Dr. Resendiz made me realize the importance of my research and how it can impact the lives of others.

Finally, I would like to mention Md Inzamam Ul Haque, Abdullah Al Bashit, and Vidya Thanda Setty, three Texas State alumni from whom I have asked for advice from time to time regarding my work. My work builds up on Inzamam's thesis, so he knew about the application of my research. Abdullah has had experience implementing machine learning with audio signal processing. Vidya has had experience with noise addition to clean speech.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In this thesis work, a robust speech emotion recognition system has been developed to be used by children with autism spectrum disorder (ASD). Children with ASD have difficulty identifying human emotions during social interactions, and the goal of this work was to develop a tool that could be used by these children to better understand the emotions of people around them. The speech emotion recognition solution was created using machine learning and deep learning techniques. A novel approach was taken, which involves joining multiple machine learning algorithms using ensemble learning to classify speech recordings in real-time. A support vector machine (SVM), a multilayer perceptron (MLP), and a recurrent neural network model were trained on the Ryerson Audio-Visual Database of Emotional Speech and Songs (RAVDESS), the Toronto Emotional Speech Set (TESS), the Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D), and a custom dataset which contains utterances from the three datasets with added background noise. Two separate audio feature sets were used, and their performances were compared. One of them was a custom feature set created specifically for this study and the other contained features from a popular speech emotion feature set. Furthermore, once the speech emotion recognizer was developed, it was joined with a facial expression recognition model to create a robust, multimodal emotion recognition system. The purpose was to get more accurate predictions of emotions by processing data from the audio and video mode.

I. INTRODUCTION

Human beings are social creatures. The concept of interdependence has been deeply rooted in society for centuries, and communication is the foundation of any community. It is through communication that valuable information is exchanged, and thoughts and feelings are shared with others. Emotional *valence* is a quantity that categorizes the different types of human emotions. Valence is measured over a spectrum, where emotions such as happiness and excitement are labeled as emotions with positive valence, while emotions like sadness and fear are considered emotions with negative valence. By detecting the emotional valence in a social interaction, the listener can take appropriate actions [1]. Like valence, arousal is another quantity that is used when categorizing emotions. The arousal measures an emotion's intensity, so emotions such as calm and boredom have a low arousal, while emotions like nervousness and excitement have high arousal. Valence and arousal form the two dimensions of emotion classification in a dimensional model, such as the one shown in Figure 1. Researchers who agree with the dimensional model of emotions believe that every emotion falls somewhere in this two-dimensional space. They believe that a standard neurophysiological system is responsible for creating all emotions. Other psychologists disagree with this categorization and think that every emotion is generated from a different neural system. In this work, this discrete categorization of emotions is used, which, by definition, excludes the measurement of valence and arousal.

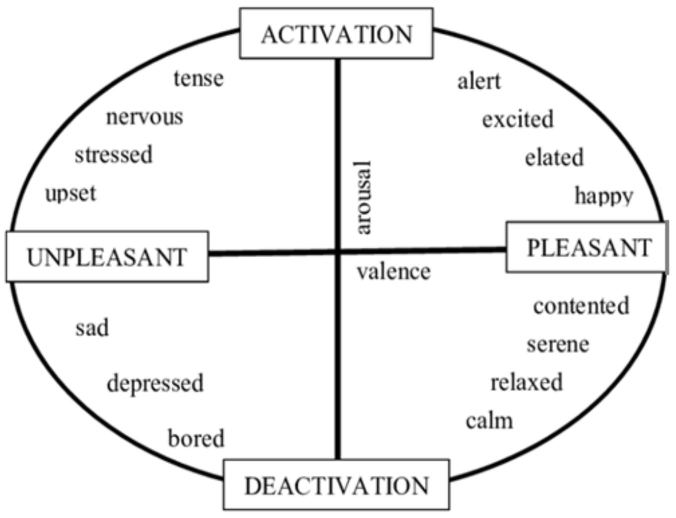


Figure 1. The Circumplex Model – a dimensional model for emotions [2].

Human interactions can be either verbal or non-verbal. Non-verbal communication has two subcategories – communication through facial expressions and communication through body language. In the year 1971, Albert Mehrabian described the *7-38-55 rule* of personal communication [3]. According to this rule, in any social interaction, 7 % of the information being conveyed comes from the spoken words, 38 % comes from the vocal tone, and 55 % comes from the speakers' body language. In that same year, the work in [4] published about the six universally recognized emotions observed throughout all cultures around the world. These emotions are happiness, sadness, surprise, anger, fear, and disgust. For the research conducted in this work, seven affective states were considered, the six emotions listed in [4] along with the neutral emotion.

The National Institute of Mental Health (NIMH) has described autism spectrum disorder (ASD) as "*a development disorder that affects communication and behavior*" [5]. For most people, understanding a person's emotions in a conversation is a simple task. However, children who fall on the autism spectrum have difficulty identifying these emotional cues - the reason is still unclear [6]. ASD can be diagnosed within the first two years of a child’s life, and doctors rely on the child’s behavior to diagnose since there are no medical tests that can be used to detect ASD as of now. Figure 2 shows all the medical conditions that are part of ASD.

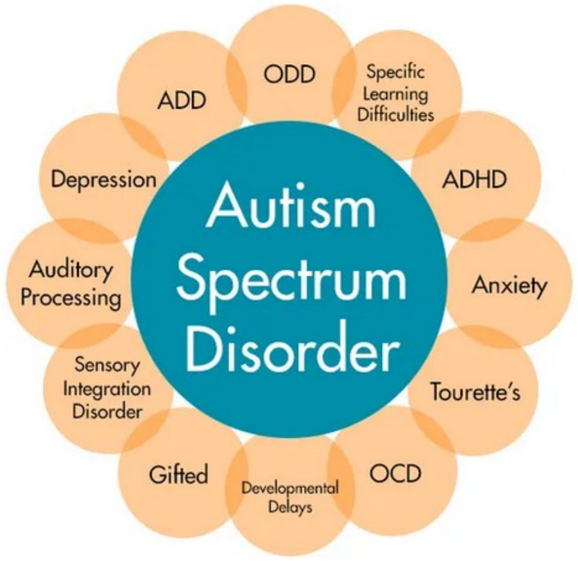


Figure 2 A list of every medical condition that is classified as autism spectrum disorder [7].

Literature Review

Speech signal processing has made significant technological advancements. Some of its applications include speaker recognition [8], automatic speech recognition [9], language recognition [10], and mental stress detection [11]. Another huge area of application is speech emotion recognition (SER). A speech emotion recognition system can be used in call centers in order to assess customer satisfaction [12], or it can be used to improve the learning experience of users in e-learning platforms [13], or it can even be used in assistive robots to make them more empathetic towards humans [14].

Machine learning techniques in speech processing have become increasingly common, thanks to massive improvements in computational power over the past three decades. A few different machine learning and deep learning classifiers have proven to yield great results for emotion speech recognition. The three most common techniques include the support vector machine (SVM), the multilayer perceptron (MLP), and the recurrent neural network (RNN). The authors in [15] used three layers of binary SVM models for the multi-class emotion recognition task. Each model was trained on one emotion and classified that emotion against the other emotions in a one-versus-all (OVA) fashion. The Interactive Emotional Dyadic Motion Capture (IEMOCAP) was used as the dataset, and features such as energy, pitch, mel-frequency cepstral coefficients (MFCC), perceptual linear predictive (PLP), filter bank, and first and second derivatives of all features were extracted as a frame-based feature using the Kaldi toolkit.

In [16], a multilayer perceptron (MLP) was trained on the Emotional Prosody Speech and Transcripts (EPST), an English speech emotion corpus, and KSUEmotions, an Arabic speech emotion corpus, to create a multi-lingual speech emotion classifier. Audio features used include pitch, intensity, formants, jitter, shimmer, and speech rate. These features were extracted using the PRAAT software package, and different combinations of these audio features were tested and compared.

The work in [17] used RECOLA, a speech emotion dataset in the French language, along with a cascaded deep learning architecture that consisted of a convolutional neural network (CNN) followed by recurrent long short-term memory (LSTM) layers. The CNN learned the audio features from raw utterances, which avoided the need for traditional hand-engineered feature extraction – a process dubbed "end-to-end speech emotion recognition."

The authors in [18] also implemented a cascaded system by studying various combinations of a support vector regression (SVR) model and a bidirectional long short-term memory deep recurrent neural network (BLSTM-DRNN). One implementation, dubbed "dependent training," used the first model's prediction output to be fed to the second model's features, along with the other audio features. The other implementation, "independent training," involved training the models separately but adding Gaussian white noise to the data used for training the first model to modify the true labels and create pseudo predictions. These pseudo predictions were then used as features for the second model along with the other audio features.

In [19], the authors used six SVM classifiers in an OVA binary classification method. All SVM classifiers used the radial basis function (RBF) kernel, and each one gave confidence of an input utterance being the emotion it was trained upon with the input samples. The final prediction came from the SVM classifier that gave the highest confidence. The LDC speech emotion corpus was used, and the final model performance was compared with naïve human coders.

The authors in [20] created a virtual game for children with ASD. The goal was to teach these children how to recognize and express emotions in a game scenario through facial expressions, tone-of-voice and body gestures. The results of the study indicated that there was an improvement in the emotion recognition and socialization skills of the participating children.

In this thesis, a speech emotion recognition system was developed using ensemble learning. Three machine learning algorithms – a SVM, an MLP, and a RNN were trained separately and combined using majority voting to give the final emotion class prediction. The datasets they were trained on include the Ryerson Audio-Visual Database of Emotional Speech and Songs (RAVDESS), the Toronto Emotional Speech Set (TESS), and the Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D). Additionally, a noise file was generated and added to the final pool of data samples used to train and evaluate the speech emotion recognition system for each clean speech utterance of the three datasets. The idea was to create a speech emotion classifier that would be impervious to environmental background noise that is present during conversations. After the speech emotion recognition model was evaluated, it was used to make predictions on speech recordings in real-time. Finally, the speech emotion recognition system was combined with a facial expression recognition system to create a multimodal emotion recognition solution for children with ASD.

II. AUDIO FEATURES

Recording Audio

Any sound is created by an object's vibration, which causes local air molecules to oscillate and produce a sound wave. Sound waves are a type of mechanical wave that requires a medium to transfer energy from one point to another. For sound waves, this medium is air. Sound waves that exist in nature are analog, continuous-time signals. It needs to be converted into a digital, discrete-time signal to record and store sound. The conversion is done by sampling the audio amplitudes at discrete points in time. The number of audio samples taken per second is defined as the *sampling rate*. Figure 3 shows the result of using different sampling rates.

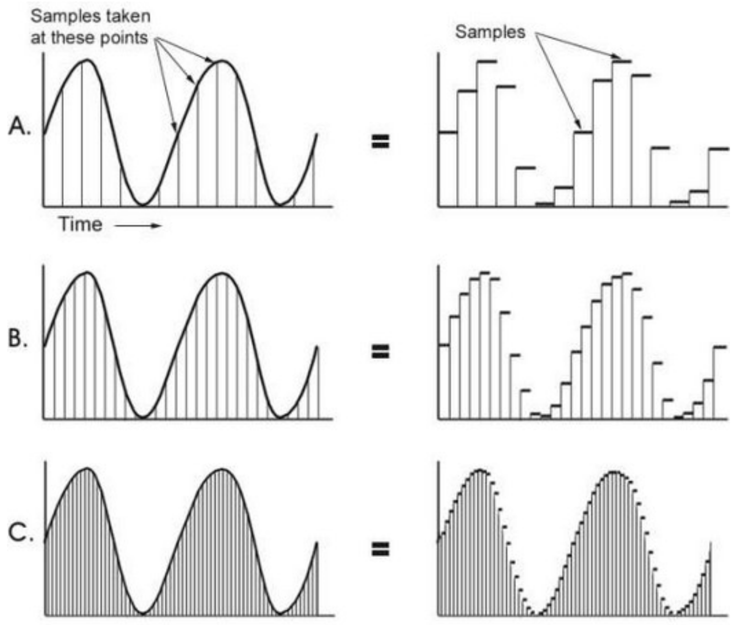


Figure 3. Increasing the sampling rate of an audio signal, where the analog continuous-time signal is shown on the left-hand side of the equal sign [21].

Even though increasing the sampling rate allows for a better approximation of the actual analog signal, it also increases data being recorded. A sampling rate of 16 kHz (16,000 samples/second) is usually used for most audio signal processing applications. Furthermore, since analog audio signals can have an infinite number of possible amplitude values, the amplitude needs to be discretized when converting into a digital signal. The *bit depth* is the number of possible amplitude values for a single sample of a discrete-time audio signal. Figure 4 shows an analog signal sampled at a bit depth of 4 bits per sample, which gives a total of 2^4 or 16 possible amplitude values for each audio sample. Like the sampling rate, the higher the bit depth, the higher the discrete-time audio signal's resolution. Most audio recordings today are 16-bit audio, with $2^{16} = 65{,}536$ possible amplitude values.

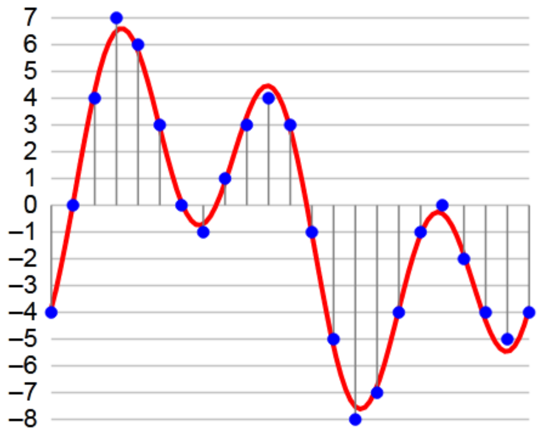


Figure 4. An analog signal (shown in red) is sampled at a bit depth of 4 bits per sample [22].

Audio Preprocessing

A speech signal is a result of a non-stationary process and therefore creates non-stationary data. This means that the statistical properties of the data, such as mean amplitude, standard deviation, and other metrics, change over time. The speech signal is divided into multiple overlapping audio segments called *frames* to make the data statistically stationary in each frame so that fast Fourier transform (FFT) can be applied for spectral analysis. A typical audio frame is somewhere between 20 to 40 milliseconds long. Figure 5 shows an example of framing a continuous-time signal.

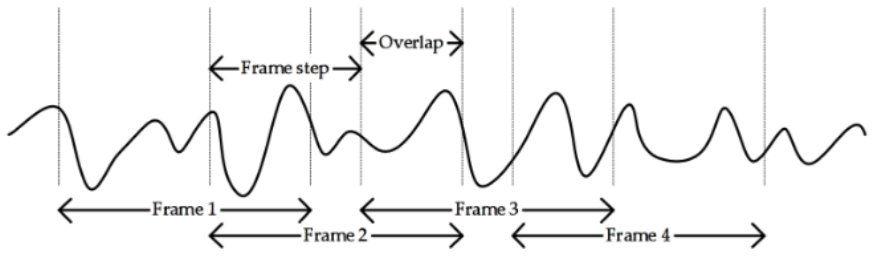


Figure 5. Framing an audio signal [23].

When FFT is applied to any signal, it is assumed that the signal is periodic. Speech signals are non-periodic by nature, and since they do not drop to zero amplitude at the end of each audio frame, the FFT will create high-frequency artifacts at these places. A window function is applied to the audio frames to avoid this issue. This technique is called *windowing*. A window function is a mathematical function that has an amplitude of zero outside some defined interval. When a window function is multiplied with the speech segment in an audio frame, the resulting speech segment will have an amplitude of zero outside the interval defined in the window function. This essentially smoothens out the edges of the signal in each audio frame. The overlapping regions in audio frames, which are around ten milliseconds long, ensure no audio segment is lost during preprocessing. Figure 6 shows an example of windowing.

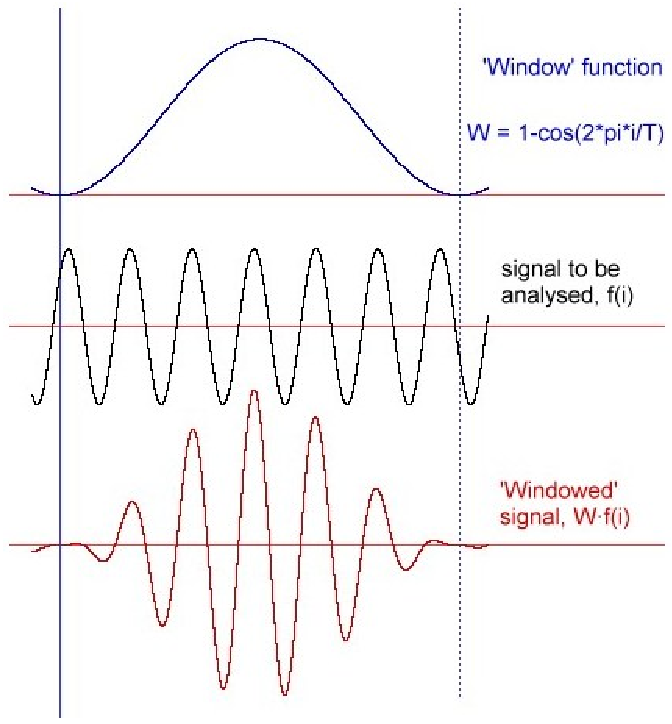


Figure 6. Using a window function on a sinusoid [24].

Features in Speech Emotion Recognition

Audio features are essential in any audio classification task, whether it be speech emotion recognition, speaker recognition, automatic speech recognition, or mental stress detection. In machine learning, features are properties of data that help a machine learn to differentiate between data classes. Researchers have used a wide variety of audio features to classify emotions in speech. In this thesis, two feature sets have been studied. Some of the common low-level descriptors used in emotion speech recognition will be briefly described in the following paragraphs.

The *mel-frequency cepstral coefficients* (*MFCCs*) were first introduced in [25]. The first step in calculating the MFCCs is to frame the audio signal into small overlapping audio frames of length 25-40 ms. For a sampling rate of 16 kHz and a frame length of 32 ms, this results in a total of 16x32 = 512 audio samples per frame. Moreover, if the frame step size (hop length) is 16 ms, it results in 16x16 = 256 audio samples in the overlapping regions. Next, the periodogram estimate of the power spectrum is calculated for each audio frame. Then, the spectrum's powers are mapped onto the *mel-scale* using a mel-filterbank that contains a set of 20-40 triangular overlapping filters, as shown in Figure 7. These filters are spaced according to the mel-scale and give the filterbank energies when applied to the power spectrum. After getting the filterbank energies, the logarithm function is applied to them. This is done because human beings do not hear loudness on a linear scale. The log operation compresses the features so that they match more closely to what humans hear. The final step is to perform a discrete cosine transform (DCT) on the log mel-filterbank energies, which gives the mel-frequency cepstral coefficients.

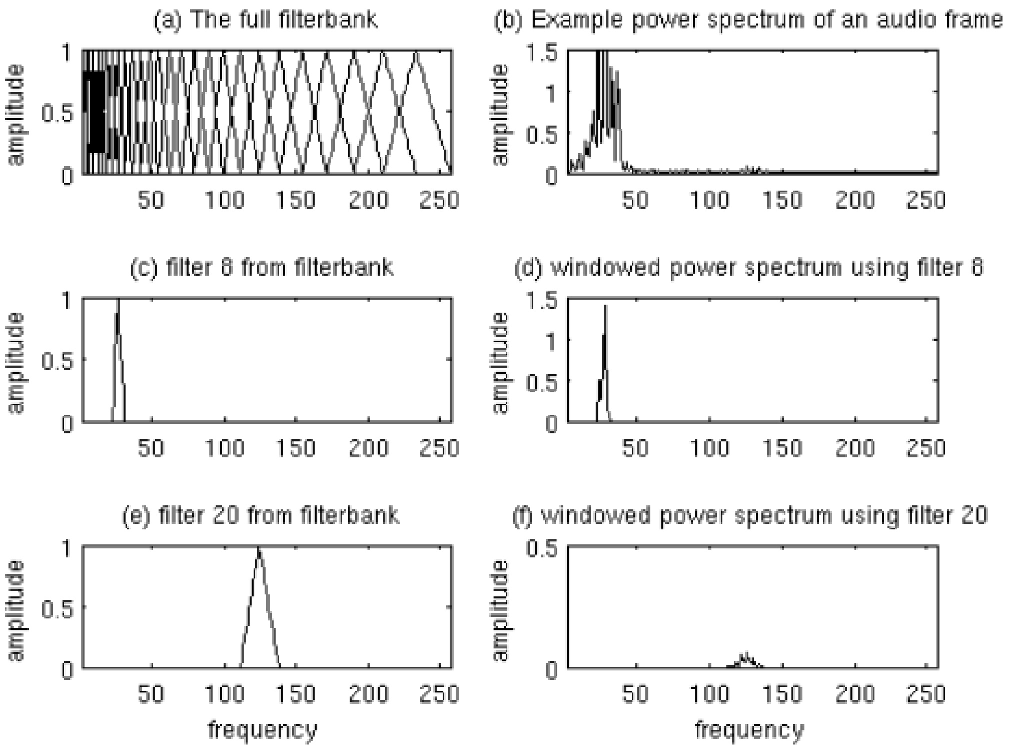


Figure 7. Applying a mel-filterbank to the power spectrum of an audio frame [26].

The *pitch* of a sound is the perceived fundamental frequency F_0, the frequency at which vocal cords vibrate in voiced sounds. Even though the pitch is a qualitative measure, for speech analysis purposes, it is considered to be equal to the logarithmic F_0 [27]. Just like pitch, *loudness* is another qualitative measure. Loudness is the perceived intensity of any sound [28].

Formants are specific peak frequencies of vocal tract resonance. They determine the quality of vowels in speech. The first three formant frequencies are labeled F_1, F_2, and F_3. Formants usually occur at 1,000 Hz intervals [29]. *Harmonics-to-noise ratio* (*HNR*) is a measure that relates the energy in the periodic part of speech (harmonics) to the energy in the noise section measured in decibels (dB) [30].

Jitter and *shimmer* are standard perturbation measures in speech analysis. Jitter is a measure of the instability of the fundamental frequency, while shimmer is a measure of amplitude instability in dB [31].

Feature Set 1: Custom

The first feature set used in this study is a custom feature set which was created specifically for this work. A trial-and-error method was used on a group of unconventional audio features. The result was a collection of 36 low-level descriptors. They are:

- *MFCCs*: The first 26 MFCCs were extracted for each audio frame using the HTK implementation [32].
- *Spectral contrast*: It represents the relative spectral distribution [33]. Seven spectral contrast values were extracted per audio frame.
- *Polynomial coefficients*: Coefficients of fitting an nth-order polynomial to the columns of a spectrogram. Two polynomial coefficients were extracted per audio frame for a polynomial of order one [34].
- *RMS energy*: The root-mean-square energy of each audio frame. One RMS energy was extracted per audio frame.

All 36 low-level descriptors mentioned above were extracted from speech data using *Librosa*, a Python library for music and audio analysis [35]. The Python library does all of the audio processing, including framing and windowing. For the final speech emotion recognition model, a sampling rate of 16 kHz was used along with a frame length of 32 ms (512 samples) and a step size of 16 ms (256 samples). For performing

FFT, 512 samples were considered per audio frame. *Librosa* is a reliable tool for audio feature extraction and has been used by researchers for various audio classification tasks. [36][37][38][39].

Functionals are functions that are applied to a vector. Examples include the mean, standard deviation, maximum, minimum, median, mode, and other metrics. Since each low-level descriptor is extracted for each audio frame, applying functionals provides feature values for the entire audio signal. For the MFCCs, the mean and the standard deviation functionals were used. The mean functional was used for the spectral contrast, polynomial coefficients, and RMS energy. This resulted in a total of 62 audio features in the custom feature set, as listed in Table 1.

Table 1. List of audio features used in the custom feature set.

Low-level Descriptors	***Functionals***	***Audio Features***
26 MFCCs	Mean, standard deviation	52
7 Spectral Contrasts	Mean	7
2 Polynomial Coefficients	Mean	3
1 RMS Energy	Mean	1

Feature Set 2: Partial GeMAPS

In [40], the authors proposed the *Geneva Minimalistic Acoustic Parameter Set* (*GeMAPS*) for affective computing. They state that there should be a standardized set of features used in speech emotion detection so that results from different publications can be compared. They also claim that large, “brute-force” feature sets lead to over-adaptation of machine learning classifiers to the training data, as the high-dimensional feature sets may cause overfitting, which leads to lower generalization capability in classification. The GeMAPS feature set contains a total of 62 audio features (including functionals of low-level descriptors). The extended GeMAPS feature set (*eGeMAPS*) contains an additional 26 audio features. The authors have compared the performance of the GeMAPS and eGEMAPS feature sets with other much larger feature sets, such as the one used in the INTERSPEECH Computational Paralinguistics Challenges (*ComParE*) [41] and found that the performances were very much comparable.

The GeMAPS feature set can be extracted using the *OpenSMILE* toolkit [42]. Since the speech emotion recognition model in this study will be deployed in the Python environment, and since there are currently no Python libraries that allow users to access the features of OpenSMILE, the GeMAPS feature set was not used in this study. However, some of the GeMAPS feature set's audio features were extracted using a Python library called *Parselmouth* [43]. It allows users to implement PRAAT features, a well-known computer software package among speech researchers [44]. By using Parselmouth, the following fifteen low-level descriptors were obtained from data:

MFCCs: The first four mel-frequency cepstral coefficients were extracted per audio frame.

Pitch: One pitch (log F_0) value was extracted per audio frame.

Loudness: One loudness (intensity) value was extracted per audio frame.

Formants: The first three formant (F_1, F_2, and F_3) frequencies were extracted per audio frame.

Formant Bandwidths: The first three formant bandwidths were extracted per audio frame.

HNR: One harmonics-to-noise ratio was extracted per audio frame.

Jitter: One local absolute jitter value was computed for an entire audio file.

Shimmer: One local shimmer value was computed for an entire audio file.

For pitch and loudness, the functionals used were the mean, the standard deviation, the 20th percentile, the 50th percentile, the 80th percentile, and the range of 20th to 80th percentile. For MFCCs, formant frequencies, formant bandwidths, and HNR, the mean and standard deviations were considered. This resulted in a total of 36 audio features in this *partial* GeMAPS feature set, as shows in Table 2. All previously mentioned settings were used for audio processing - sampling rate of 16 kHz, frame length of 32 ms (512 samples), step size of 16 ms (256 samples), and FFT of size 512 samples per audio frame.

Table 2. List of audio features used in the partial GeMAPS feature set.

Low-level Descriptors	*Functionals*	*Audio Features*
1 Pitch	Mean, standard deviation, 20th, 50th, 80th percentile, range of 20th to 80th percentile	6
1 Loudness	Mean, standard deviation, 20th, 50th, 80th percentile, range of 20th to 80th percentile	6
4 MFCCs	Mean, standard deviation	8
3 Formants	Mean, standard deviation	6
3 Formants BWs	Mean, standard deviation	6
1 HNR	Mean, standard deviation	2
1 Jitter	Mean	1
1 Shimmer	Mean	1

III. DATASETS

For this research work, three separate speech corpora were selected. They are the Ryerson Audio-Visual Database of Emotional Speech and Songs (RAVDESS), the Toronto Emotional Speech Set (TESS), and the Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D). All three datasets are available online to be used by researchers, free of cost. These datasets were designed and created specifically for speech emotion recognition and were evaluated and validated by multiple individuals. There are two main types of speech datasets that are used by researchers in this field. The first type contains recordings of people who express genuine emotions by being subjected to external influence, such as image, video, and audio; the second type contains recordings of professional actors reading outlines from a script while acting out the emotions. The former type of dataset is known in the literature as a spontaneous dataset, and the latter is known as a simulated dataset. Researchers usually use simulated speech data for speech emotion recognition task because they are accurately labeled, and actors are very good at expressing each emotion with reasonable accuracy. Also, most spontaneous datasets are limited in terms of the number of emotion classes.

Out of the three speech corpora used, the RAVDESS and CREMA-D corpora contain multi-modal audio and video data. The thesis objective is to develop an emotion classifier for children who are North American English speakers, and all three datasets selected for this work contain recordings of actors for whom English is the first language. Furthermore, the datasets chosen had to include the same seven emotion classes used by that model as the final speech model is to be integrated with the High-Performance Engineering (HiPE) research group's facial expression recognition model [45].

Ryerson Audio-Visual Database of Emotional Speech and Songs

The Ryerson Audio-Visual Database of Emotional Speech and Songs (RAVDESS) was released in 2018 by researchers of the SMART lab at Ryerson University in Toronto, Ontario, Canada. It is a simulated, multi-modal dataset that contains both video data and audio data. For the audio data, the actors recorded the sentences both as normal speech and as songs. The song data was not considered for this thesis. The audio files were recorded at 16 bits per sample, at a sampling rate of 48 kHz, and in WAV audio format. A total of 24 actors of age range 21-33 years had taken part in creating this dataset, where half of the samples contain male actors and the other half are female actors. Each actor recorded two lexically matched sentences in eight different emotions. The two sentences are “Kids are talking by the door” and “Dogs are sitting by the door.” Moreover, the eight emotions are neutral, calm, happy, sad, anger, fear, disgust, and surprise. The emotion labels are included in the WAV audio file names. Out of the eight emotions, seven of them were recorded twice per sentence – once with normal intensity and the other time with stronger intensity. There was only one recording per sentence for the neutral emotion since there is no strong intensity for this emotion. Since the HiPE group’s facial expression recognition model was not trained on any calm emotion data, this class was excluded from the study. This gives a total of 1,440 recording samples, with 24 actors x 2 sentences x 8 emotions x 2 repetitions x 2 emotional intensity with the exception of the neutral emotion. Thus, seven emotions have 192 data samples, and the neutral class has 96 data samples. Data resampling was used to match the neutral class count to the rest of the classes [46]. The RAVDESS speech corpus can be downloaded from [47].

Toronto Emotional Speech Set

The second speech corpus used for this study was the Toronto Emotional Speech Set (TESS). This simulated dataset was created in 2010 by researchers from the University of Toronto Psychology Department. It contains recordings from two actors, both females. The younger actor was 26 years old at the time of recording, while the older actor was 64 years old. They have recorded 2,800 sentences in seven different emotions – anger, disgust, fear, happy, surprise, sad, and neutral. Unlike RAVDESS, this is a balanced dataset with each emotion class having 400 data samples. However, just like RAVDESS, the sentences spoken by the actors are lexically similar. Each actor recorded the phrase "Say the word___" followed by one of 200 different target words in each affective state. All audio files were recorded at 16-bits per sample, at a sampling rate of 24,414 Hz, and saved in WAV audio file format. The emotion labels were extracted from the file names [48]. The TESS dataset can be downloaded from [49].

Crowd-sourced Emotional Multimodal Actors Dataset

The Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D) was the third speech emotion dataset used to develop the speech model in this thesis. Just like RAVDESS and TESS, this is also a simulated speech corpus. It was released in 2014 as a collaboration between researchers from the University of Pennsylvania, Ursinus College and the University of Illinois at Chicago. Actors who participated had ages ranging from 20 to 74 years old. There were 48 male actors and 43 female actors with a total of 91, and even though they came from different ethnic backgrounds, they were all English speakers. Like RAVDESS, it is also a multimodal dataset. The speech recordings were done at 16-bits per sample, at a sampling rate of 16 kHz, and saved in the WAV audio format. The actors recorded twelve different emotionally neutral sentences in six different emotions of anger, disgust, fear, happy, neutral, and sad, at four different intensity levels of low, medium, high, and unspecified. Examples phrases include “Don’t forget a jacket,” “I think I’ve seen this before,” “I think I have a doctor’s appointment.” Each emotion class has 1,271 data samples, except for the neutral class, which has 1,087 data samples. Resampling was used to create a balanced dataset [50]. The CREMA-D dataset can be accessed from [51]. Table 3 gives a summary of the three datasets used for this thesis work.

Table 3. Summary of all three speech emotion corpora

Corpus	***Age of participants***	***No. of sentences***	***No. of participants***	***Emotions***	***Samples per class (balanced)***	***Total (balanced)***
RAVDESS	21-33 years old	2 (with 2 repetitions & intensities)	24 (12 Males, 12 Females)	8 (calm excluded)	192	1,344 (192 x 7)
TESS	26 and 64 years old	200 (3 common words, 1 changing)	2 (0 Males, 2 Females)	7	400	2,800 (400 x 7)
CREMA-D	20-74 years old	12 (four intensities)	91 (48 Males, 43 Females)	6 (surprise missing)	1,271	7,626 (1,271 x 6)

Noise Addition

All speech recordings in the above-mentioned datasets were done in the absence of background noise, in a "noiseless" environment. The only noise that is audible in these recordings is a combination of the inherent static noise created by the recording equipment, and the echoes coming from the surroundings. In real-world applications, however, conversations in a "noiseless" environment are very rare because there is always some noise around people during a conversation in different environments. Therefore, it was important to take these background noises into consideration when constructing the final speech emotion recognition model.

The problem with training a machine learning model with "noiseless" data, or clean speech data, is that it trains and performs well when tested. However, in real-world applications, it will struggle to identify the emotions accurately because of undesired contributions of surrounding noises to the recorded audio. A new set of data was created from the clean speech recordings by introducing some background noise to them to address the background noises. Figure 8 shows the noise addition process, where a city center noise recording was added to a clean speech recording from the RAVDESS dataset. The RAVDESS recording is of actor ID:22 saying "Dogs are sitting by the door" with strong surprise emotion.

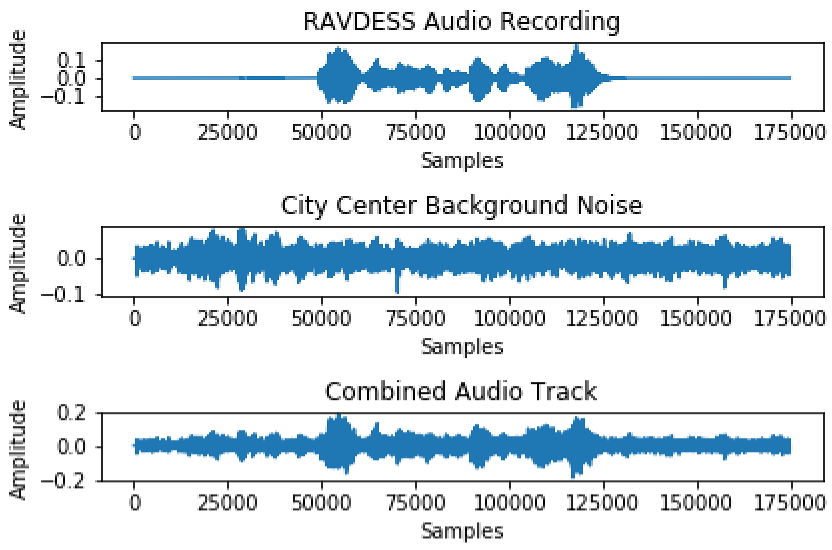


Figure 8. Waveforms of a RAVDESS utterance (top), a noise sample (middle), and their combination (bottom).

It is impossible to consider all possible types of background noises when creating the new dataset with added background noise. Therefore, three noise samples were selected for the purpose of this study. All three audio files were downloaded from the same website [52]. The first noise sample is a recording of children playing in a playground titled "Small Crowd", the second noise sample is a recording of a shopping mall titled "Shopping Mall Ambiance", and the third noise sample is a recording of cars passing by on the streets titled "Street". The playground noise sample and the street noise sample were listed under an attribution 3.0 license, while the shopping mall noise sample was listed under a public domain license. The reason behind selecting these specific noise samples was to cover some of the general properties of common background noises. For example, the playground background noise is a variation of the cocktail party noise where the noise created is from people talking in the background. Also, as the speech emotion classifier is being built for children with ASD, it is expected that they will be in the presence of other children in a similar setting. The shopping mall noise was selected because it has an echo effect created by the high ceilings which is also observed in some other environments such as airports and churches. Finally, the street noise was selected because the sound of cars is a common background noise outdoors.

After these three noise samples were selected, three signal-to-noise ratios (SNRs) were selected: 0 dB, 5 dB, and 10 dB. These SNRs were used to add the noise samples to the clean speech data. Noise was added to all data samples in all three datasets using the three noise types and three SNR values, as follows. This noise addition procedure was carried out using MATLAB. By randomly picking an SNR value out of the three and randomly picking a noise type out of the three, the noise was added to each data sample. To prevent the machine from learning the noise features, two precautions were taken. First, each noise sample is more than fifteen seconds long, with the shopping mall noise 16-seconds long, playground noise 19-seconds long, and street noise 49 seconds long. The clean speech samples were roughly three to five seconds long for all datasets. The MATLAB code randomly took chunks of noise audio samples the size of the clean speech samples and added both together to create the noise-added samples. This ensured that the same part of the noise samples was not being added to the clean speech data. Second, also, for each clean speech sample, only one noise-added sample was generated, where noise type and SNR were randomly chosen as discussed above. Thus, the final dataset contained the same amounts of clean speech and noise-added samples.

Combining all three datasets without balancing the classes with utterances of 1,248 from RAVDESS, 2,800 from TESS, and 7,442 from CREMA-D, 11,490 more data samples were created by noise addition. The resultant final dataset contains 22,980 audio data samples. However, the final count of the dataset after resampling the neutral and surprise emotion classes from RAVDESS and CREMA-D is 26,082. Table 4 summarizes the details of the final dataset.

Table 4. Summary of the final dataset

Corpus	***Data Samples***
RAVDESS	1,248
TESS	2,800
CREMA-D	7,442
Combined (RAVDESS + TESS + CREMA-D)	11,490
Noise-added (RAVDESS + TESS + CREMA-D)	11,490
Final (Combined + Noise-added)	22,980
Final (balanced)	26,082

IV. MACHINE LEARNING ALGORITHMS

Machine learning is the science of teaching computer algorithms on how to make decisions without human's involvement. It falls under the broader field that is artificial intelligence. In *supervised* machine learning, a *learning* algorithm is exposed to labeled data and learns the different data categories by matching the features to the labeled data samples. Once the algorithm has been trained on the data, it can make predictions on new, unseen data. The term 'machine' in machine learning refers to the computer that contains the learning algorithm. *Deep learning* is a subsection of machine learning that refers to the use of advanced neural network architectures with multiple layers of neurons. Classification and regression are the two branches of supervised learning. Classification is performed on discrete data, while regression is performed on continuous data. There are several types of learning algorithms in supervised learning. Each of these algorithms is applied to different types of data. For instance, the convolutional neural network (CNN) is designed to work like the human brain's visual cortex and is therefore solely used in image processing applications. A few algorithms have shown great performances with audio signal processing, more specifically, in speech emotion recognition tasks. In this work, three of these algorithms will be studied – the support vector machine (SVM), the multilayer perceptron (MLP), and the recurrent neural network (RNN). All three supervised learning algorithms have been used in this thesis to perform the classification of seven discrete affective states.

Support Vector Machine

The support vector machine (SVM) is a supervised machine learning algorithm created by V. N. Vapnik and his colleagues at AT&T Bell Laboratories [53]. It can be considered as an extension to the perceptron learning algorithm. Figure 9 shows how an SVM classifies data samples in a binary classification problem.

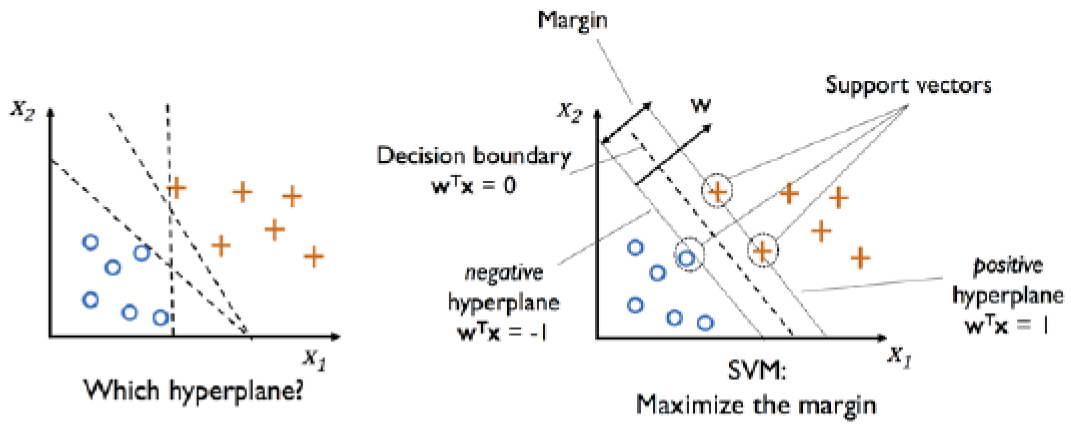


Figure 9. Plots showing the working principle of the support vector machine algorithm [54].

In the plots, the crosses and circle signs represent the data samples of two different outcome classes. The dotted line separating the data samples is called a hyperplane, and the data samples closest to the hyperplane are called the support vectors. Even though there are many possible ways to linearly separate the classes, as shown in the left-hand plot of Figure 9, the SVM algorithm creates a maximum-margin hyperplane, such that it maximizes the margin between the support vectors. This margin is the distance between the two parallel hyperplanes shown as solid lines in the right-hand plot of Figure 9. One is the positive hyperplane and the other is the negative hyperplane. The equations for these hyperplanes are given below.

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{pos} = 1 \ (1)$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{neg} = -1 \ (2)$$

In Equations (1) and (2), the term w_0 is the weight of the bias term, the term $\boldsymbol{w^T}$ indicates the transposed weight vector. Assuming that the two classes have labels +1 for the positive class, shown as "+", and -1 for the negative class, shown as "o" in Figure 9. The $\boldsymbol{x_{pos}}$ and $\boldsymbol{x_{neg}}$ are the input feature vectors of the positive class and the negative class, respectively. The bold terms represent the vector dot product operation. Subtracting Equation (2) from (1) gives,

$$\boldsymbol{w}^T(\boldsymbol{x_{pos}} - \boldsymbol{x_{neg}}) = 2 \text{ (3)}$$

The length of the vector $\boldsymbol{w}$ can be calculated as,

$$||\boldsymbol{w}|| = \sqrt{\sum_{j=1}^{m} w_j^2} \text{ (4)}$$

Normalizing Equation (3) by dividing both sides by $||\boldsymbol{w}||$ yields the following equation,

$$\frac{\boldsymbol{w}^T(\boldsymbol{x_{pos}} - \boldsymbol{x_{neg}})}{||\boldsymbol{w}||} = \frac{2}{||\boldsymbol{w}||} \text{ (5)}$$

The left side of Equation (5) can be interpreted as the margin or distance between the positive and negative hyperplane. This margin will be maximum if the term on the right of the equal sign of Equation (5) is maximum. This maximization must be performed while correctly classifying the data samples. Therefore, for $i = 0...N$, where N is the total number of samples in the dataset,

$$w_0 + \boldsymbol{w}^T\boldsymbol{x}^{(i)} \geq 1 \; if \; y^{(i)} = 1 \text{ (6)}$$

$$w_0 + \boldsymbol{w}^T\boldsymbol{x}^{(i)} \leq -1 \; if \; y^{(i)} = -1 \text{ (7)}$$

Equation (6) means that if the true label $y^{(i)}$ of a data sample is +1 (the positive class), it must fall above the positive hyperplane. Similarly, Equation (7) means that if the true label $y^{(i)}$ of a data sample is -1 (the negative class), it must fall below the negative hyperplane. The hyperplane right at the middle of the two parallel hyperplanes, shown as a dotted line in the right-hand plot of Figure 9, is the optimal, maximum-margin hyperplane calculated by the SVM algorithm under the constraints defined by Equations. (6) and (7). Maximizing the margin is the objective function of the SVM algorithm. In practice, instead of maximizing the term $2/||\boldsymbol{w}||$ of Equation (5), it is more convenient to minimize its reciprocal ½*$||\boldsymbol{w}||^2$.

Vapnik came up with the soft-margin classification to relax the linear constraints defined in Equations (6) and (7) for nonlinearly separable data. He introduced a slack variable ξ, which can be added to Equations (6) and (7),

$$w_0 + \boldsymbol{w}^T\boldsymbol{x}^{(i)} \geq 1 - \xi^{(i)}\ if\ y^{(i)} = 1 \ (8)$$

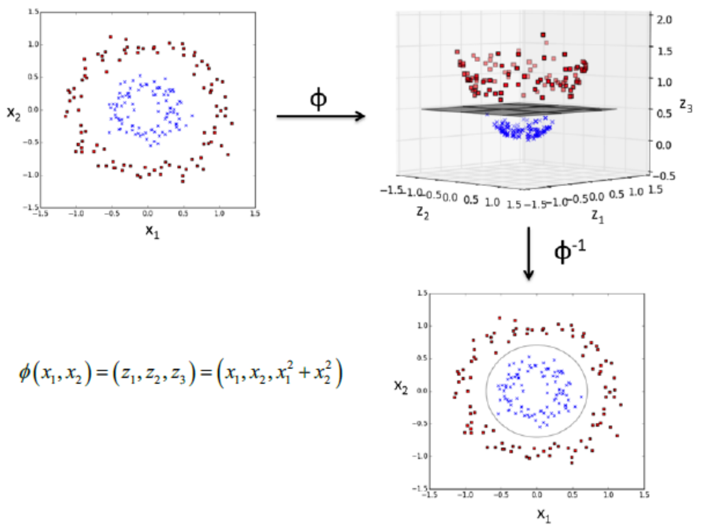
$$w_0 + \boldsymbol{w}^T\boldsymbol{x}^{(i)} \leq -1 + \xi^{(i)}\ if\ y^{(i)} = -1 \ (9)$$

Now, the following term needs to be minimized to maximize the margin,

$$\frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + C\left(\sum_i \xi^{(i)}\right) \ (10)$$

Here, the *C* parameter is the penalty variable which the users can tweak. Increasing the value of *C* means increasing the error penalty, while decreasing its value means being more lenient when punishing the model for misclassification error.

There are some problems where the classes are not linearly separable. In cases non-linearity solutions, the SVM can be kernelized and add dimensionality for the proper hyperplane utilize supporting vectors for maximum separation. The basic idea of kernel SVM is to create nonlinear combination of the input features $\boldsymbol{x}$ to project the features onto a higher-dimensional space. This in turn allows for the data samples to be linearly separable by defining a hyperplane using the linear SVM algorithm. A mapping function ϕ is used to create higher-dimensional features from the input features. Figure 10 depicts the so-called "kernel trick" by using a mapping transformation to generate the hyperplane boundary and inverse transform to represent the classification boundary to the original sample space.



x_2
x_1
ϕ
z_3
z_2
z_1
ϕ^{-1}
$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$
x_2
x_1

Figure 10. A visual representation of mapping input features x_1 and x_2 to a higher-dimensional space, allowing the kernel SVM to produce the hyperplane required to separate the two classes [54].

The radial basis function (RBF) is a very popular kernel function, and it is represented by,

$$\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp(-\gamma \left|\left|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right|\right|^2) \quad (11)$$

The SVM is one of the most widely used classifiers in machine learning due to its ability to work with multiple classes. It has also been extensively used in the field of emotion speech recognition and has yielded great results. That is why the SVM was selected as one of the machine learning classifiers in this study. In Equation (11), the term γ (gamma) is a free parameter that can be optimized by the user. For this thesis work, the C parameter and the γ parameter were tuned for the RBF kernel SVM.

Multilayer Perceptron

The Perceptron

Artificial neurons were inspired by the real biological neurons of the human body. Neurons, or nerve cells, are the building blocks of the human nervous system. Figure 11 shows the body of a typical neuron. The dendrites collect electrical and chemical signals, synapses, from other neurons, which are combined at the nucleus. If the aggregate signal exceeds a threshold, the neuron fires a synapse through its axon to other neurons.

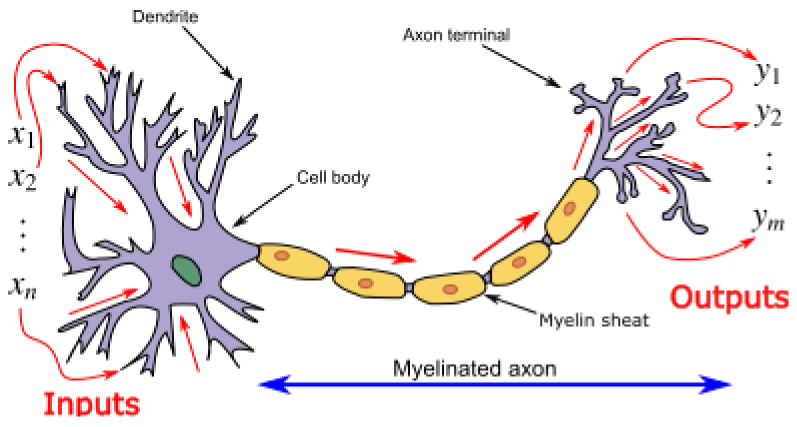


Figure 11. The structure of a biological neuron [55].

In 1943, Warren McCulloch and Walter Pitts came up with the concept of the McCulloch-Pitts (MCP) neuron [56]. They described the MCP neuron to be a simple logic gate with a binary output. Just a few years later based on the same original concept, Frank Rosenblatt published about the perceptron learning algorithm [57]. According to Rosenblatt, the perceptron would automatically learn the optimal weights and then multiply them with their respective input features. It would then add the numbers and determine if the outcome reaches a threshold level to fire the neuron or not. This logical process mechanism is used in supervised learning to predict either one of more classes.

For a binary classification problem, supervised learning, where the two classes are labeled +1 for the positive class or -1 for the negative class, there will be a set of input features. Since each input feature $\boldsymbol{x_i}$ has a corresponding weight coefficient $\boldsymbol{w_i}$ so there will be two vectors, as shown below,

$$\boldsymbol{w} = [w_1 \ldots . w_m] \ (12)$$

$$\boldsymbol{x} = [x_1 \ldots . x_m] \ (13)$$

For classification, if the net input of a data sample $\boldsymbol{x^{(i)}}$ is bigger than a threshold θ, the sample is classified as the positive class (+1); otherwise, the sample is predicted to be from the negative class (-1). This decision is made by the decision function, $\phi()$, which is defined in Equation (14).

$$\phi(z) = \begin{cases} 1 \; if \; z \geq \theta \\ -1 \; otherwise \end{cases} \ (14)$$

Bringing the threshold θ to the left side of the equal sign for simplifying Equation (14) creates an additional term $\boldsymbol{w_0 x_0}$, where $\boldsymbol{w_0} = -\theta$ and $\boldsymbol{x_0} = 1$. Thus, Equation (14) becomes:

$$\phi(z) = \begin{cases} 1 \; if \; z \geq 0 \\ -1 \; otherwise \end{cases} \ (15)$$

One of the vectors needs to be transposed to calculate the net input of a data sample, which is a vector dot product between $\boldsymbol{w}$ and $\boldsymbol{x}$. Transposing $\boldsymbol{w}$ gives $\boldsymbol{w^T}$. Therefore, the net input can be expressed as,

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x} \ (16)$$

The perceptron learning rule is quite simple. At first, the feature weights are initialized to either random values or simply zeroes. Then, for each input training sample the output is calculated. Then, based on the error calculated, all the weights are updated simultaneously before the next training sample is fed to the perceptron. The weight vector can be written as,

$$w_j := w_j + \Delta w_j \quad (17)$$

The term Δw_j is the weight update and is determined by:

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)} \quad (18)$$

In Equation (18), $\boldsymbol{y^{(i)}}$ is the true label of the $\boldsymbol{i}$**th** training sample, $\boldsymbol{\hat{y}^{(i)}}$ is the predicted label, and η is the learning rate that has a floating-point number between 0.0 and 1.0. The learning rate determines how fast the algorithm learns the features of the training data. A high learning rate might cause the model to overshoot the global minimum of the loss function, while a low learning rate might take a long time to converge to a minimum loss. Figure 12 summarizes the idea behind the perceptron learning rule.

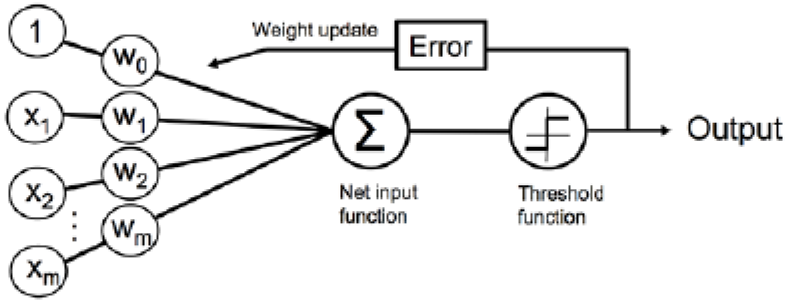


Figure 12. The perceptron learning rule [54].

Feedforward Artificial Neural Networks

The multilayer perceptron (MLP) is a feedforward artificial neural network consisting of multiple artificial neurons arranged into an input layer, one or more hidden layers, and an output layer. Figure 13 shows the architecture of a single hidden-layered

MLP. It shows a network with one input layer, one hidden layer, and one output layer. Neurons in each layer, shown as circles, are fully connected to the ones in the next layer by weight coefficients. Each neuron is an activation unit with an activation function, where $\boldsymbol{a_i^{(in)}}$ refers to the ***i*th** value in the input layer. The units $\boldsymbol{a_0^{(in)}}$ and $\boldsymbol{a_0^{(h)}}$ are the bias units set to 1. The weight coefficient $\boldsymbol{w_{0,1}^{(h)}}$ connects the ***0*th** unit of the ***h*** layer, or the 1st hidden layer, to the *1*st unit of the *h+1* layer, or the 2nd hidden layer.

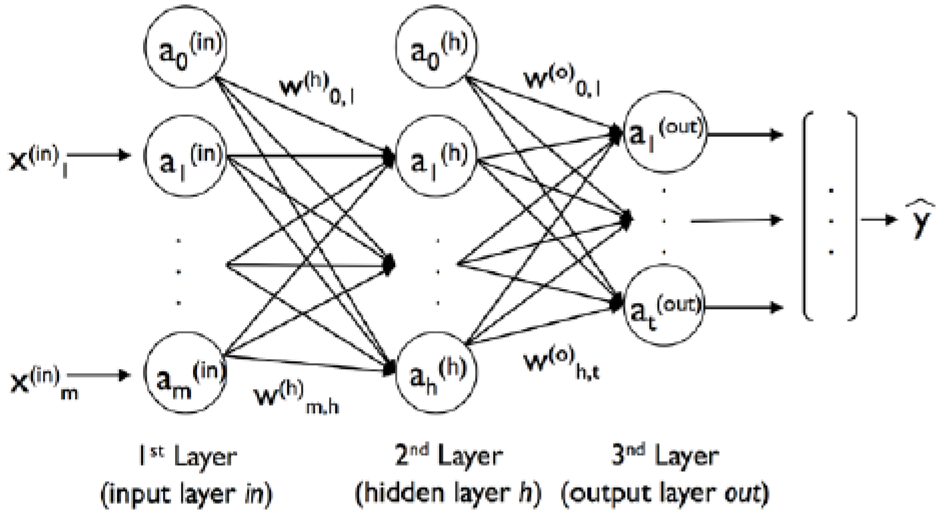
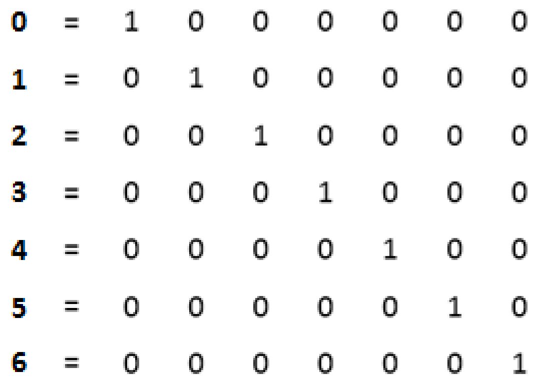


Figure 13. Structure of a typical multilayer perceptron [54].

An MLP has three main steps to its learning procedure. From the input layer, the patterns of the training data are moved forward, forward propagation, until an output is calculated by the network. Then, the error is calculated using a cost function. The output labels need to be one-hot encoded, which essentially converts the labels into a binary representation using zeros and ones, creating a vector for each label. Without one-hot encoding, the machine will assume that the integer classes have a natural order (hierarchy). Figure 14 shows how class labels are one-hot encoded. The final step is to propagate the error through the weights in the network and calculating its derivative with respect to each weight and updating the model.



0	=	1	0	0	0	0	0	0
1	=	0	1	0	0	0	0	0
2	=	0	0	1	0	0	0	0
3	=	0	0	0	1	0	0	0
4	=	0	0	0	0	1	0	0
5	=	0	0	0	0	0	1	0
6	=	0	0	0	0	0	0	1

Figure 14. One-hot encoding of integer labels.

The net output of the first hidden layer of Figure 13 can be calculated by using Equation (19).

$$z_1^{(h)} = a_0^{(in)} w_{0,1}^{(h)} + a_1^{(in)} w_{1,1}^{(h)} + \cdots + a_m^{(in)} w_{m,1}^{(h)} \ (19)$$

, where,

$$a^{(in)} = \left[a_0^{(in)} a_1^{(in)} \ldots a_m^{(in)}\right] = \left[1 \ x_1^{(in)} \ldots x_m^{(in)}\right] (20)$$

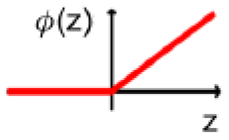
Thus, the first activation unit of the first hidden layer can be calculated as,

$$a_1^{(h)} = \phi(z_1^{(h)}) (21)$$

Activation Function

The function $\phi()$ is the notation for the activation function. It must be differentiable to learn the weights using gradient descent. A non-linear activation function is used to solve complex problems, such as the sigmoid function, tanh function, exponential function, or other potential functions. The Rectified Linear Unit (ReLU) is a non-linear activation function that is often used in *deep neural networks*. Any network with two or more hidden layers is called a deep neural network. The ReLU activation has the following equation,

$$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$$



(22)

Optimization

The gradient descent is an optimization technique used in machine learning. Optimization is the process of framing a problem in order to maximize/minimize some goal or objective. In this case, the objective, or objective function, to be minimized is the cost function, or loss function $J(\boldsymbol{w})$, where $\boldsymbol{w}$ is the weight vector. Figure 15 illustrates the process of gradient descent. The goal is to reach the minimum point of the curve, which represents the weight value with the lowest cost.

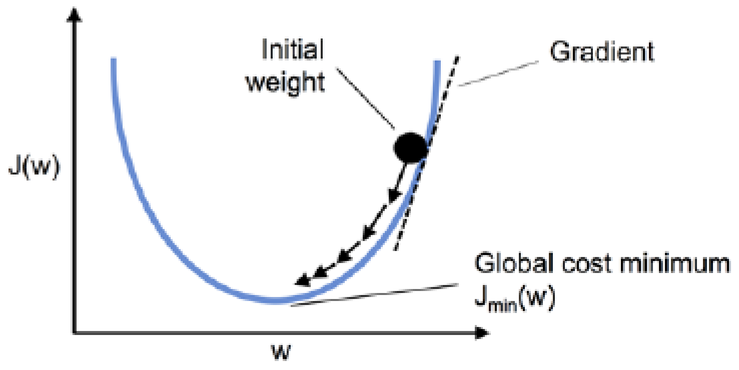


Figure 15. Minimizing the cost function using gradient descent [54].

The new weight is calculated by subtracting the step size from the old weight (weights are percentages). This is done for each individual weight of the network. The step size is given as,

$$Step\ size = Gradient\ of\ Error \times \eta \quad (23)$$

The error gradient is the sum of the partial derivatives of the cost function with respect to each weight. The error gradient becomes smaller and smaller the more layers are added to a network. For some non-linear activation functions, such as the tanh function, it might cause the error gradient to become very close to zero resulting in a slow learning process during training. This is known as the “vanishing gradient” problem. Using ReLU bypasses this issue since the derivative of ReLU with respect to its input is always 1 for positive input values.

Backpropagation

Backpropagation is the optimization technique used in artificial neural networks. For artificial neural networks, the cost function does not have a smooth convex shape as shown in Figure 16, but rather a rough surface with bumps, as shown in Figure 16. The goal is to avoid the local minimum and reach the global minimum.

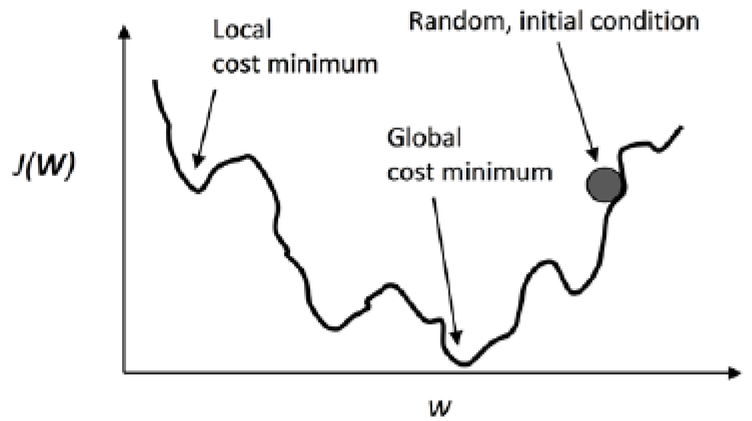


Figure 16. Cost function plot of an artificial neural network [54].

The network output is calculated using forward propagation through the dot-product calculations and predictions decision by the Activation function. The backpropagation technique is used to move the computed error backwards – from right to left of the neural network. The first step is to calculate the error vector,

$$\boldsymbol{\delta}^{(out)} = \boldsymbol{a}^{(out)} - \boldsymbol{y} (24)$$

Here, $\boldsymbol{y}$ is the vector containing true labels. Then, the error term of the hidden layer is calculated. The operator in Equation (25) indicates element-wise multiplication, and the term to the right of the operator is the derivative of the activation function.

$$\boldsymbol{\delta}^{(h)} = \boldsymbol{\delta}^{(out)}\left(\boldsymbol{W}^{(out)}\right)^{T} \odot \frac{\partial \phi(z^{(h)})}{\partial z^{(h)}} (25)$$

Next, the derivatives of the cost function with respect to the weights are calculated for every node of each layer.

$$\frac{\partial}{\partial w_{i,j}^{(out)}} J(\boldsymbol{W}) = a_j^{(h)} \delta_i^{(out)} \ (26)$$

$$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\boldsymbol{W}) = a_j^{(in)} \delta_i^{(h)} \ (27)$$

The partial derivate for each node and the error of the node in the next layer is aggregated. This gives the error gradient since the gradient is the sum of all partial derivatives.

$$\Delta^{(h)} = \mathit{\Delta}^{(h)} + \left(\boldsymbol{A}^{(in)}\right)^{T} \delta^{(h)} \ (28)$$

$$\Delta^{(out)} = \mathit{\Delta}^{(out)} + \left(\boldsymbol{A}^{(h)}\right)^{T} \delta^{(out)} \ (29)$$

Then, the regularization term is added to the error gradient for layer *l*.

$$\Delta^{(i)} := \mathit{\Delta}^{(i)} + \lambda^{(i)} (except\ for\ the\ bias\ term) \ (30)$$

Finally, the weights are updated by taking an opposite step towards the gradient for each layer *l*.

$$\boldsymbol{W}^{(l)} := \boldsymbol{W}^{(l)} - \eta \Delta^{(l)} \quad (31)$$

The entire backpropagation method is depicted in Figure 17. The neurons with 1's are the bias units.

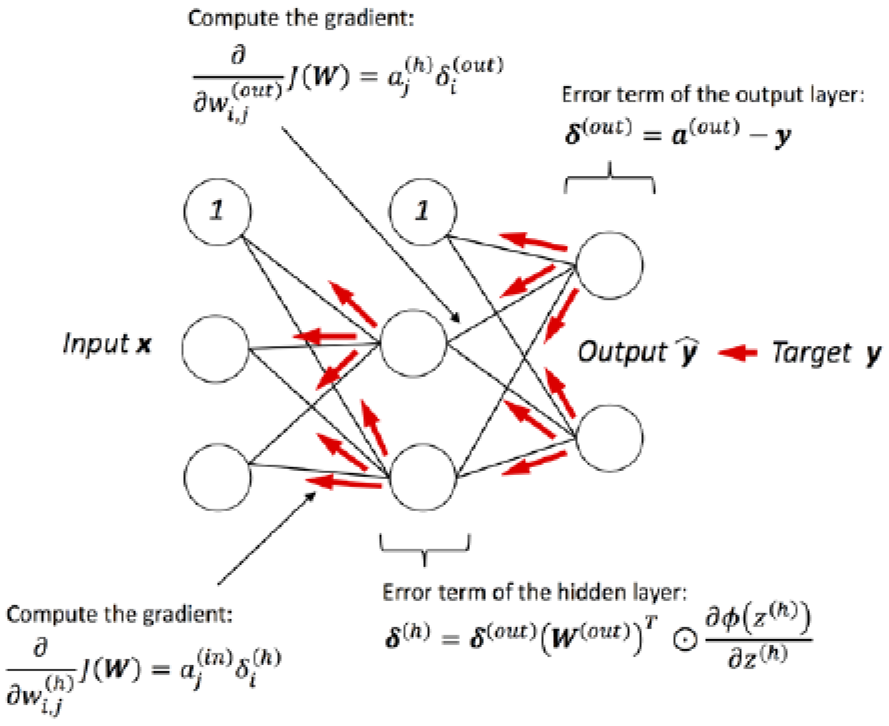

Compute the gradient:
$\frac{\partial}{\partial w_{i,j}^{(out)}} J(\boldsymbol{W}) = a_j^{(h)} \delta_i^{(out)}$
Error term of the output layer:
$\boldsymbol{\delta}^{(out)} = \boldsymbol{a}^{(out)} - \boldsymbol{y}$
1
1
Input **x**
Output $\hat{y}$
Target **y**
Compute the gradient:
$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\boldsymbol{W}) = a_j^{(in)} \delta_i^{(h)}$
Error term of the hidden layer:
$\boldsymbol{\delta}^{(h)} = \boldsymbol{\delta}^{(out)} \left(\boldsymbol{W}^{(out)}\right)^T \odot \frac{\partial \phi\left(z^{(h)}\right)}{\partial z^{(h)}}$

Figure 17. Error moving from right to left in backpropagation [54].

For this thesis, a variety of hyperparameters of the MLP were experimented and tested for accuracy performance and evaluation. The list includes the network architecture in the number of neurons and layers, activation unit, optimizer, cost or loss function, epochs, batch size, learning rates, regularization, and dropout.

Recurrent Neural Network

Sequences

Recurrent neural networks (RNNs) are a special class of artificial neural networks that are used with sequential data. RNN is a type of *deep learning* algorithm. The data used in the training process is assumed to be independent and identically distributed (IID), meaning that the order in which they are fed to the algorithm under the supervised learning approach is not relevant. However, the order of the input data matters for sequential data, or *sequences*.

Standard neural network models such as the multilayer perceptron (MLP) or the convolutional neural network (CNN) cannot process ordered input data samples since these networks do not take past information into consideration during training RNNs are designed specifically for sequences because the technique can remember information from the previous data samples and learn the sequential patterns of the data. One common example of sequences is a time-series data, where each data sample $x^{(t)}$ belongs to a specific time t.

Structure of an RNN

In a feedforward neural network like an MLP, the flow of information is from the input layer to the hidden layer(s), and then finally to the output layer for the calculated predicted values. In RNNs, the input to a hidden layer comes from both the input layer and the hidden layer from the previous time step. A simplified representation is demonstrated in Figure 18.

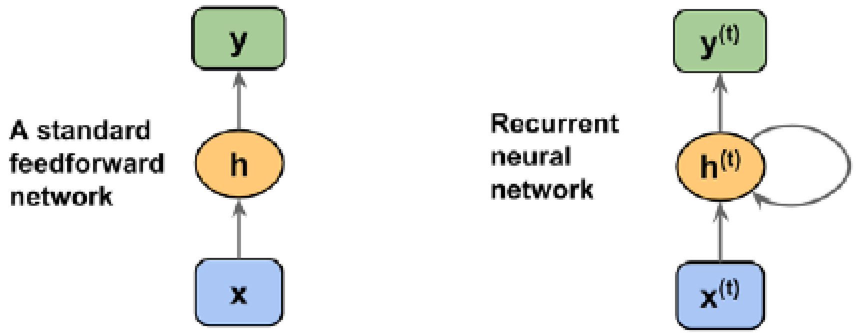


Figure 18. Comparing architectures of an MLP and an RNN with one hidden layer [54].

In Figure 18, *x* represents the input layer, *h* represents the hidden layer, and *y* represents the output layer. The looped arrow in the hidden layer *h* for the RNN structure is called the *recurrent edge*. It represents the flow of information in the hidden layer, from the previous time step *t-1* to the current time step *t*. The units, artificial neurons, in each layer are not shown, but it is assumed that each layer is a vector of multiple neurons. Figure 19 shows the inner workings of a single layer RNN and a multilayer RNN. In both cases, each hidden layer receives two inputs – the pre-activation, z-input, from the input layer, and the activation of the same hidden layer but from the previous time step *t-1*.

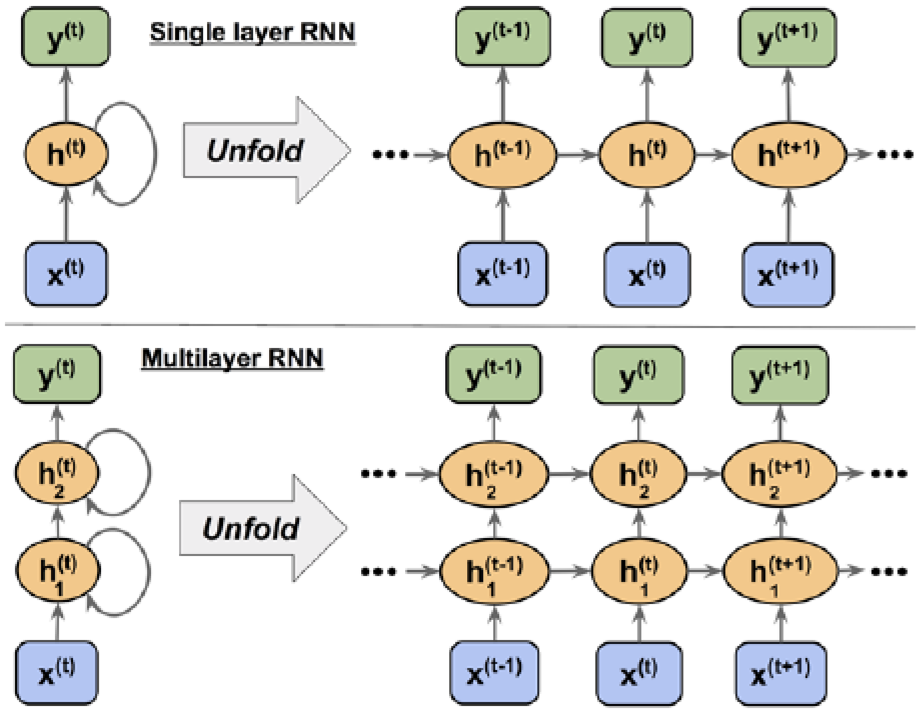


Figure 19. Single layer RNN and multilayer RNN [54].

Learning Challenges of RNNs

RNNs learn using a technique described in [54] called backpropagation through time (BPTT). The loss L is defined as the sum of all the loss functions at times t=1 through t=T, where T is the last time step.

$$L = \sum_{t=1}^{T} L^{(t)} \quad (32)$$

The loss at time 1:t depends on the hidden units of all previous time steps, 1:t because there are long-range dependencies associated with sequences. The gradient of the loss is computed as follows,

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_{hh}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{y}^{(t)}} \times \frac{\partial \boldsymbol{y}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \times \left(\sum_{k=1}^{t} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(k)}} \times \frac{\partial \boldsymbol{h}^{(k)}}{\partial \boldsymbol{W}_{hh}}\right) \quad (33)$$

, where the multiplicative factor $\partial \boldsymbol{h}^{(t)}/\partial \boldsymbol{h}^{(k)}$ in Equation (33) is computed as a multiplication of adjacent time steps and has t-k multiplications. Thus, multiplying the weight t-k times results in w^{t-k}. As a result, if $|w_{hh}|<1$, w^{t-k} becomes very small when t-k is large.

This causes the *vanishing gradient* problem. This creates the *exploding gradient* problem when $|w_{hh}|>1$, w^{t-k} becomes very large when t-k is large. Therefore, these problems can be avoided if $|w_{hh}|=1$. In practice, the long short-term memory (LSTM) network is the most popular technique to avoid the vanishing gradient problem.

Long Short-Term Memory

LSTMs were first introduced in 1997 [58]. The LSTM is made up of memory cells that are equivalent to hidden layer units. Each memory cell has a recurrent edge with weight $|w|=1$, which prevents the vanishing gradient and exploding gradient problem. The value associated with this recurrent edge is called the cell state C^k. Thus, along with a hidden state, h^k, each LSTM cell maintains a cell state C^k. And unlike standard RNN cells which have one activation unit per cell, the LSTM has four activation units per cell. Figure 20 shows the structure of an LSTM cell.

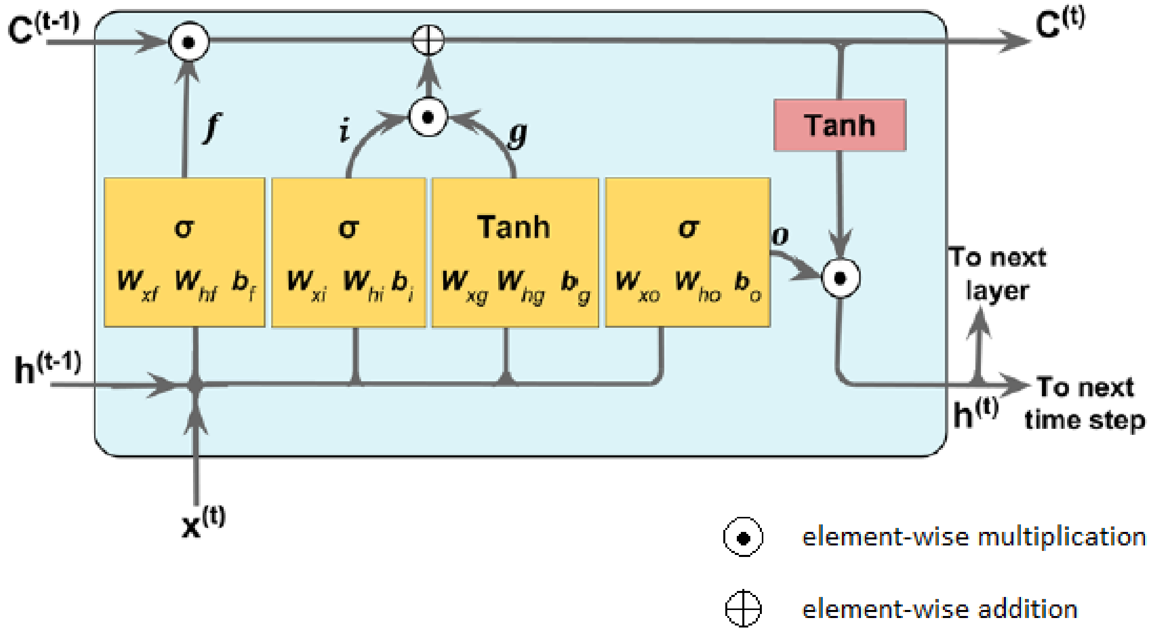


Figure 20. Unfolded LSTM cell [54].

The cell state in the current time step $C^{(t)}$ is calculated by modifying the cell state from the previous time step $C^{(t\text{-}1)}$ without multiplying with any weight factors. In Figure 20, $\boldsymbol{x}^{(t)}$ is the input at time t, and $\boldsymbol{h}^{(t\text{-}1)}$ represents the hidden units at time t-1. The four yellow boxes contain an activation function along with a set of weights. The boxes with the sigmoid activation function (σ) are called *gates*. In an LSTM cell, there are three types of gates:

1. The *forget gate* ($\boldsymbol{f}_t$) suppresses irrelevant information while letting useful information go through. This allows the cell to reset its state to zero without growing indefinitely. Equation (34) shows how the forget gate vector, $\boldsymbol{f}_t$, is calculated for an entire LSTM layer. The input weight vector of the forget gate, $\boldsymbol{W}_{xf}$, is multiplied with the current input vector, $\boldsymbol{x}^{(t)}$, and the hidden weight vector of the forget gate, $\boldsymbol{W}_{hf}$, is multiplied with the previous hidden state vector, $\boldsymbol{h}^{(t\text{-}1)}$. Then, the results are added with the bias vector of the

forget gate, $\boldsymbol{b}_f$, and the summation is passed through a sigmoid activation function, σ, in order to get the output vector of the forget gate, $\boldsymbol{f}_t$.

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{xf}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hf}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f) \quad (34)$$

2. The *input gate* ($\boldsymbol{i}_t$) and the input node ($\boldsymbol{g}_t$) work together and update the cell state. Equation (35) shows how the input gate vector, $\boldsymbol{i}_t$, is calculated for an entire LSTM layer, and Equation (36) shows how the input node vector, $\boldsymbol{g}_t$, is calculated for an entire LSTM layer. The input weight vector of the input gate, $\boldsymbol{W}_{xi}$, is multiplied with the current input vector, $\boldsymbol{x}^{(t)}$, and the hidden weight vector of the input gate, $\boldsymbol{W}_{hi}$, is multiplied with the previous hidden state vector, $\boldsymbol{h}^{(t-1)}$. Then, the results are added with the bias vector of the input gate, $\boldsymbol{b}_i$, and the summation is passed through a sigmoid activation function, σ, in order to get the output vector of the input gate, $\boldsymbol{i}_t$. Similarly, the input weight vector of the input node, $\boldsymbol{W}_{xg}$, is multiplied with the current input vector, $\boldsymbol{x}^{(t)}$, and the hidden weight vector of the input node, $\boldsymbol{W}_{hg}$, is multiplied with the previous hidden state vector, $\boldsymbol{h}^{(t-1)}$. Then, the results are added with the bias vector of the input node, $\boldsymbol{b}_g$, and the summation is passed through a hyperbolic tangent activation function, tanh, in order to get output vector of the input node, $\boldsymbol{g}_t$.

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{xi}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hi}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i) \quad (35)$$

$$\boldsymbol{g}_t = \tanh(\boldsymbol{W}_{xg}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hg}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_g) (36)$$

, $\boldsymbol{C}^{(t)}$, the current cell state vector for the entire LSTM layer is computed as shown in Equation (37). An element-wise multiplication is performed between the previous cell state vector, $\boldsymbol{C}^{(t-1)}$, and the output vector of the forget gate, $\boldsymbol{f}_t$. Another element-wise multiplication is performed between the output vector of the input gate, $\boldsymbol{i}_t$, and the output vector of the input node, $\boldsymbol{g}_t$. The resulting vector elements are added by an element-wise summation operation.

$$\boldsymbol{C}^{(t)} = (\boldsymbol{C}^{(t-1)} \odot \boldsymbol{f}_t) \oplus (\boldsymbol{i}_t \odot \boldsymbol{g}_t)(37)$$

3. The *output gate* ($\boldsymbol{o}_t$) updates the values of the hidden units. The output gate vector, $\boldsymbol{o}_t$, for an entire LSTM layer is calculated as shown in Equation (38). The input weight vector of the output gate, $\boldsymbol{W}_{xo}$**,** is multiplied with the current input vector, $\boldsymbol{x}^{(t)}$, and the hidden weight vector of the output gate, $\boldsymbol{W}_{ho}$, is multiplied with the previous hidden state vector, $\boldsymbol{h}^{(t-1)}$. Then, the results are added with the bias vector of the output gate, $\boldsymbol{b}_o$, and the summation is passed through a sigmoid activation function, σ, in order to get the output vector of the output gate, $\boldsymbol{o}_t$.

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{xo}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{ho}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o) \ (38)$$

, $\boldsymbol{h}^{(t)}$, the hidden state vector at the current time steps for an entire LSTM layer is calculated as shown in Equation (39). The current cell state vector, $\boldsymbol{C}^{(t)}$, is passed through a hyperbolic tangent activation function, tanh. The resulting vector is multiplied with the output vector of the output gate, $\boldsymbol{o}_t$, by an element-wise multiplication operation.

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}^{(t)}) \ (39)$$

Just like the MLP model, a list of different RNN hyperparameters were tuned for getting the best results. The list includes the network architecture for the number of neurons and layers, activation unit, optimizer, cost or loss function, batch size, learning rates, regularization, and dropout.

V. RESEARCH METHODOLOGY

Experimental Procedure

Three speech emotion corpora were collected from the internet - the Ryerson Audio-Visual Database of Emotional Speech and Songs (RAVDESS), the Toronto Emotional Speech Set (TESS), the Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D) to create the speech emotion recognition system. At first, the RAVDESS corpus was selected for training and optimizing the support vector machine (SVM), the multilayer perceptron (MLP), and the recurrent neural network (RNN) model. The RAVDESS dataset contains data from all the seven emotion classes used in this work, and there is an equal number of male and female actors. Also, the recording quality in RAVDESS is better compared to CREMA-D. Furthermore, training the models on a single dataset was faster than training them on all three. All these factors made RAVDESS an excellent first choice. However, after using the RAVDESS hyperparameter settings on the other two datasets, the results were inferior. This was because the RAVDESS dataset had very few data, and so the machine had low generalization capability when tested on other datasets. For this reason, the model tuning strategy was changed to a new strategy described below.

Three different noise samples were used in order to modify all three speech emotion corpora. This was done to train the models on speech data in the presence of noise. Most everyday conversations happen with some noise in the background. The characteristics of the background noise depend on the surrounding environment of the speakers. If the speech emotion system is only trained on clean speech data, it will not perform well in real-world applications because the system will pick up noise along with the speech signal and try to process the audio with the added noise. The audio features extracted from the audio will be misleading as they will contain components of the noise. This will eventually lead to low classification accuracies. Therefore, the models were trained and evaluated with datasets containing background noise to create a robust speech emotion recognizer. The three background noises selected are the sound of children playing in a playground, the ambiance in a shopping mall, and the sound of cars passing by on the streets. These three noise samples represent three completely different scenarios. Three different SNR values were selected for adding these noise samples to the clean speech – 0 dB, 5 dB, and 10 dB – which introduces a lot more variety to the original clean speech datasets. Different sections of the noise files were added to different clean speech files to avoid teaching the background noise's machine features during training. From RAVDESS, TESS, and CREMA-D, each clean speech file was combined with one of the three noise samples in one of the three SNRs to create a noise-added file.

Thus, instead of training the three machine learning algorithms (SVM, MLP, and RNN) on only the RAVDESS dataset, they were trained on a bigger dataset that contains the RAVDESS recordings (clean speech), the TESS recordings (clean speech), and the CREMA-D recordings (clean speech), along with the noise-added versions of these clean-speech files. A special naming convention is used from this point forward to simplify referencing these different datasets. This naming convention is explained in Table 5. The neutral class in RAVDESS and CREMA-D had fewer data samples than the other classes, so it was resampled to match the other classes. The surprise class was missing in CREMA-D, so it was resampled when all three datasets were combined.

Table 5. Dataset naming convention followed in this research.

Name	*Description*	***Data Samples** (**balanced**)*
RAVDESS_Clean	Original RAVDESS corpus	1,344
RAVDESS_Clean_Noise	Original RAVDESS corpus along with the noise-added versions	2,688
TESS_Clean	Original TESS corpus	2,800
TESS_Clean_Noise	Original TESS corpus along with the noise-added versions	5,600
CREMA-D_Clean	Original CREMA-D corpus	7,626
CREMA-D_Clean_Noise	Original CREMA-D corpus along with the noise-added versions	15,252
Complete_Clean	Original RAVDESS, TESS, and CREMA-D corpora	13,041
Complete_Clean_Noise	Original RAVDESS, TESS, and CREMA-D corpora along with the noise-added versions	26,082

Performance Metrics

Individual performance metrics have been selected to assess the results of the experiments conducted in this research. They are listed below.

Training, Validation, and Test Accuracy

In machine learning, the dataset for the problem being worked on is initially split into two parts – the *training set* and the *test set*. The training set is the part of the dataset used for training the machine learning algorithm, while the test set is the part of the dataset used to get an unbiased evaluation of the model performance. The test set is separated from the model during the training process. There is another partition of the dataset that comes from the training set. It is called the *validation set*, and like the test set, it is also separated from the model during training. Figure 21 shows how data partitioning is done in machine learning. Common ratios for the partitions (training : validation : test) are 70:10:20, 75:10:15, and 80:10:10. For this thesis, all datasets were split in the 80:10:10 ratio, with a 80% of data being reserved for training while the remaining 20% being equally split for the validation and test set. This was done to use most of the data for training. Also, all three splits were *stratified*, meaning that there were equal number of data samples per emotion class within a split.

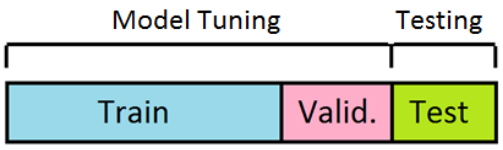


Figure 21. Splitting a dataset into the training set, the validation set, and the test set.

The formula for calculating the *training accuracy* is given in Equation 40. The *true labels* are the ground truths, i.e., the actual labels of data classes included in the dataset. The machine's number of correct predictions is simply calculated by comparing the predicted labels to the true labels. The training accuracy measures how well the machine performed when predicting samples from the training set. For computing the training accuracy, the machine makes predictions on the data it was trained on; the training accuracy is usually the highest among the three classification accuracies.

$$Training\ Acc.(\%) = \frac{No.of\ correct\ predictions\ on\ training\ samples}{Total\ no.of\ data\ samples\ in\ training\ set} \times 100 (40)$$

The *validation accuracy* is computed using Equation (41). It measures the model performance on previously unseen data by dividing the total number of correct predictions made on the validation set data with the data samples' total number in the validation set.

$$Validation\ Acc.(\%) = \frac{No.of\ correct\ predictions\ on\ validation\ samples}{Total\ no.of\ data\ samples\ in\ validation\ set} \times 100\ (41)$$

The *test accuracy* is given by Equation (42). Just like the validation accuracy, it measures how the model performs on previously unseen data. The main difference between the test accuracy and the validation accuracy is that after getting the validation accuracy score, the model can be re-tuned if the score is low or a big gap between training accuracy and validation accuracy (*overfitting*). On the other hand, the test accuracy is only computed once, after all the model parameters have been finalized. If a model is constantly re-tuned using the test accuracy instead of the validation accuracy, it will cause that model to overfit to the test data. The test accuracy is therefore, an unbiased representation of model performance.

$$Test\ Acc.(\%) = \frac{No.of\ correct\ predictions\ on\ test\ samples}{Total\ no.of\ data\ samples\ in\ test\ set} \times 100\ (42)$$

These three classification accuracies are the most popular performance metrics for machine learning and deep learning. A good machine learning model has high scores for all three classification accuracies, indicating a strong prediction capability. Furthermore, the model has almost similar validation and test accuracies and does not have a large gap between the training accuracy and validation accuracy.

Precision and Recall

Even though the classification accuracy gives an idea of how well the model performs, it does not indicate how the model is performing when classifying individual classes. The metrics of *precision* and *recall* become relevant in evaluating different classification rates. The average precision score is calculated by dividing the sum of true positives across all classes by the sum of true positives and false positives. For example, when classifying the happy emotion, the true positives are the data samples that were correctly classified as "happy." The false positives are the data samples classified as "happy," but were, in fact, one of the other six emotions. The false negatives are the "happy" data samples that were misclassified as other emotions. Finally, the true negatives are the data samples that were correctly classified as other emotions. Equation 43 shows the average precision score base on the true positive predictions over the aggregate of the true and false positive sample space.

$$Precision = \frac{True\ positives}{True\ positives + False\ positives} \quad (43)$$

The average recall score is calculated by dividing the sum of true positives by the sum of true positives and false negatives across all classes. Equation 44 summarizes this computation of the recall statistical metric.

$$Recall = \frac{True\ positives}{True\ positives + False\ negatives} \quad (44)$$

Therefore, the precision quantifies the number of correct predictions made on the test set out of all the predictions labeled by the machine as "true," while the recall quantifies the number of correct predictions made on the test set out of all the actual "true" instances. These two metrics are beneficial for classification problems, especially when the classes are not balanced. In this work, the average precision and recall scores were calculated for models making predictions on the test set.

Learning Curves

In *learning curves*, the training accuracy and validation accuracy are plotted against the number of training samples. Figure 22 shows some examples of learning curves. The vertical axis (*y-axis*) represents the classification accuracy, while the horizontal axis (*x-axis*) represents the number of training samples used for training the model. Learning curves provide a visual cue for whether a model is *underfitting* or *overfitting*. In machine learning, the *bias* error is the error caused by the learning algorithm's wrong assumptions, and the *variance* error is the error caused by the model being over-sensitive to the small changes in the training set [59]. If a model is not complex enough to learn the data's properties, it is said to have high bias and will fail to correctly classify the inputs, which will result in inadequate training and validation accuracy. This condition is called underfitting and can be rectified by increasing the number of parameters used in the model or decreasing the degree of regularization (error penalty). Figure 22(a) shows an example of an underfitted model.

In contrast, if too many parameters are being used in training a model, that model will overly complicate the learning process and adapt to the training set. This results in high variance, and the model fails to generalize when introduced to previously unseen data from the validation set. This condition is called overfitting, and it can be corrected by either using more training data, reducing the number of parameters, or increasing the strength of regularization used. Overfitting is depicted in Figure 22(b). Figure 22(c) shows learning curves with a good balance between bias and variance.

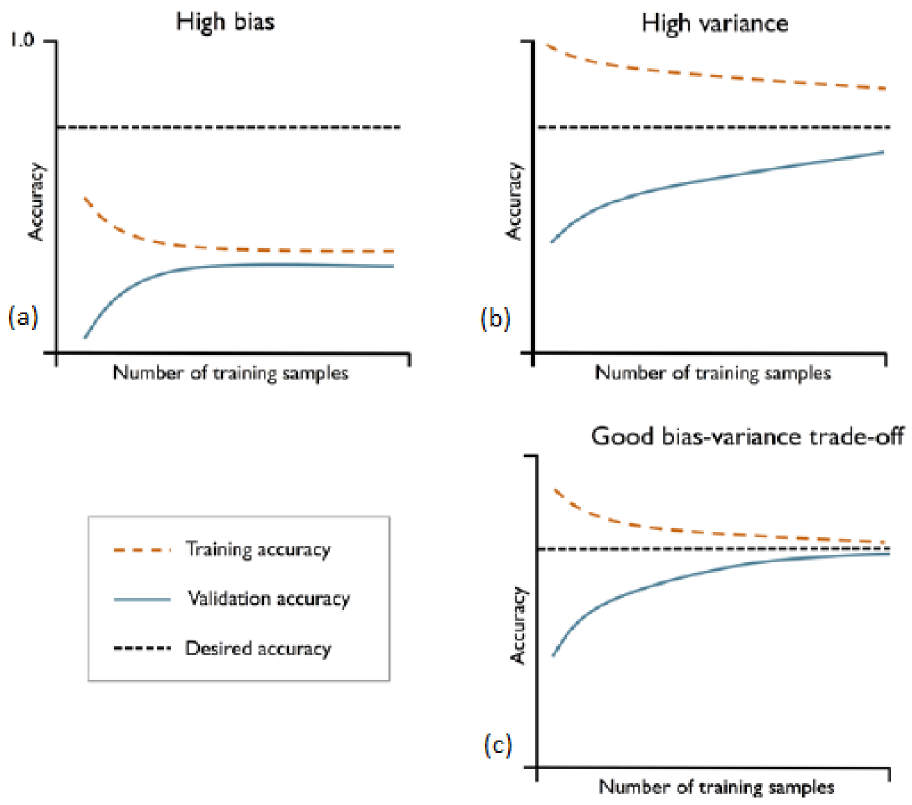


Figure 22. Learning curves showing (a) underfitting, (b) overfitting, and (c) good bias-variance trade-off [54].

Accuracy Curves

Another group of plots used to detect underfitting and overfitting are *accuracy curves*. They are very similar to learning curves, with the only difference being the quantity represented on the horizontal axis (*x-axis*). For accuracy curves, the horizontal axis shows the number of epochs, i.e., the number of passes over the training set. For example, if a model is trained over the entire training set ten times, the model is trained with ten epochs. Since artificial neural networks are trained using multiple epochs, these types of curves are useful for the MLP and RNN models used in this work. Figure 23. shows an example of accuracy curves for a well-performing model.

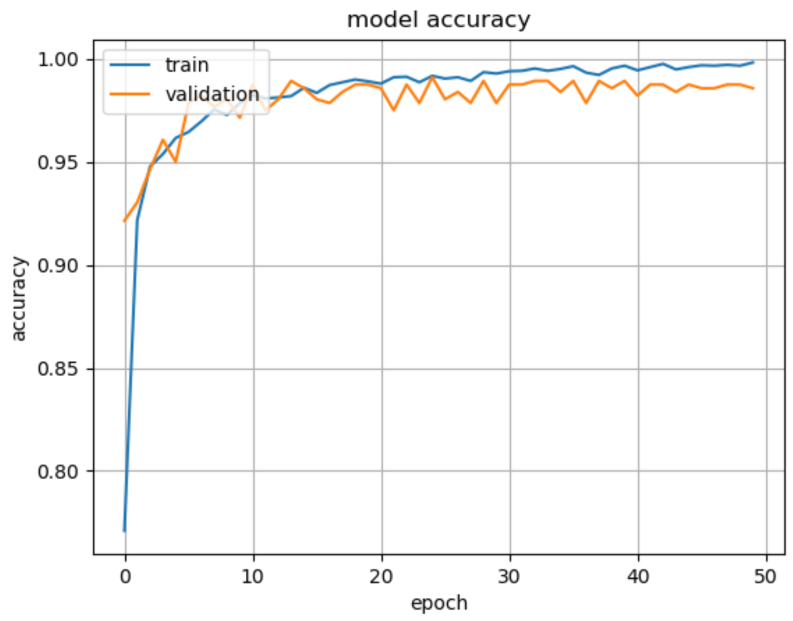


Figure 23. Accuracy curves, showing the training and validation accuracies against the number of epochs.

Loss Curves

Loss curves are plots of the training and validation losses against the number of epochs used in model training. Like accuracy curves, loss curves are used to evaluate the MLP and RNN models' performance in this study. If the validation loss keeps increasing with the number of epochs instead of decreasing, as seen in Figure 24, it indicates overfitting. The categorical cross-entropy loss is a loss function used in multiclass classification problems. This loss function calculates the cross-entropy loss between the class labels and the predictions made by the machine. In this work, the categorical cross-entropy loss is the cost function that was minimized for both the MLP and the RNN model.

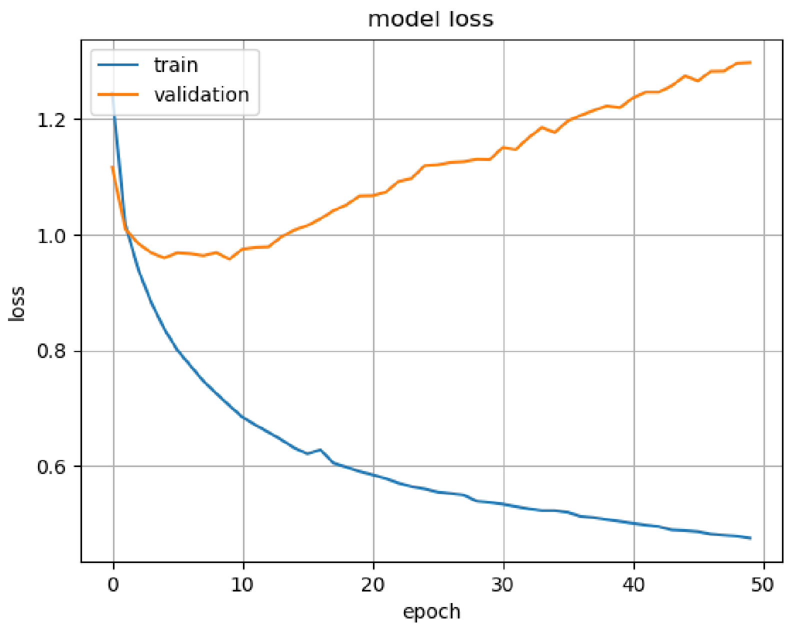


Figure 24. Loss curves, showing the case of an overfitted model.

Confusion Matrix

A *confusion matrix* is a two-dimensional array of numbers. One of its axes represents the true labels of the validation data, while the other axis represents the predicted labels of the same data samples. Figure 25 demonstrates a typical confusion matrix. In this example, the vertical axis is for the true labels, and the horizontal axis is for the predicted labels. All seven classes are labeled on both axes. The diagonal numbers (from the top left corner to bottom right corner) represent the two axes' labels. These numbers represent the number of data samples that the machine was able to correctly classify because, for these numbers, the predicted labels match the true labels. All other numbers outside this diagonal represent the number of data samples misclassified by the machine. The total number of data samples belonging to any class in the validation set can be found by adding all the numbers on a straight line corresponding to that label on the true label axis (in this case, all numbers in a row). The confusion matrix gives an idea of how well the machine is predicting data from each class.

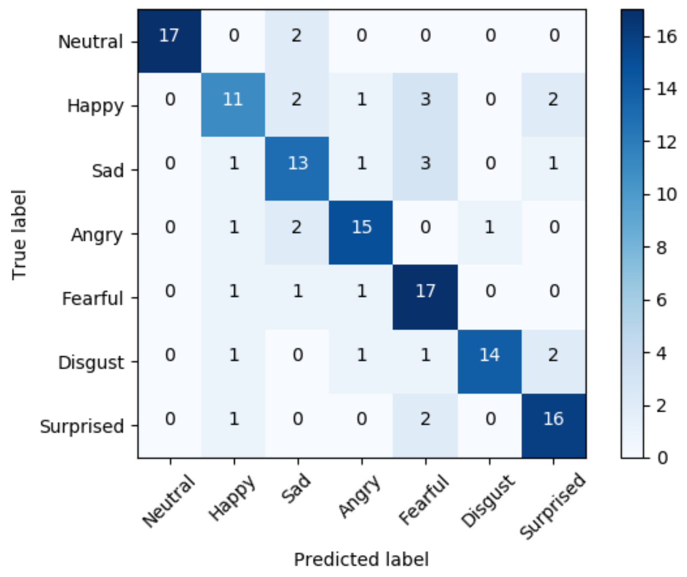


Figure 25. A confusion matrix for a dataset with seven classes.

K-fold Cross-validation

In this study, all model hyperparameters were tuned by using the same data shuffle. This was done by setting the *random_state* variable in Python to zero. Doing this was to get the same output scores for running the same program multiple times. This way, when tuning a certain hyperparameter, different values of that hyperparameter were easily compared to see which gave the best results for the same data samples. Shuffling the data for each different value of the hyperparameter would have made the comparison invalid since each model would have been trained, validated, and tested on different data samples. Another way of tuning hyperparameters is by using *k-fold cross-validation* [54]. In this technique, the training data is split into *k* folds (or sections), where *k*-1 folds are used as the training set while the remaining fold is used as the validation set. The model is trained and validated a total of *k* times, each using a different fold for the validation set. Figure 26 shows the concept of a 10-fold cross-validation. The final performance score, depicted as *E* in Figure 26, is the average score for all ten experiments.

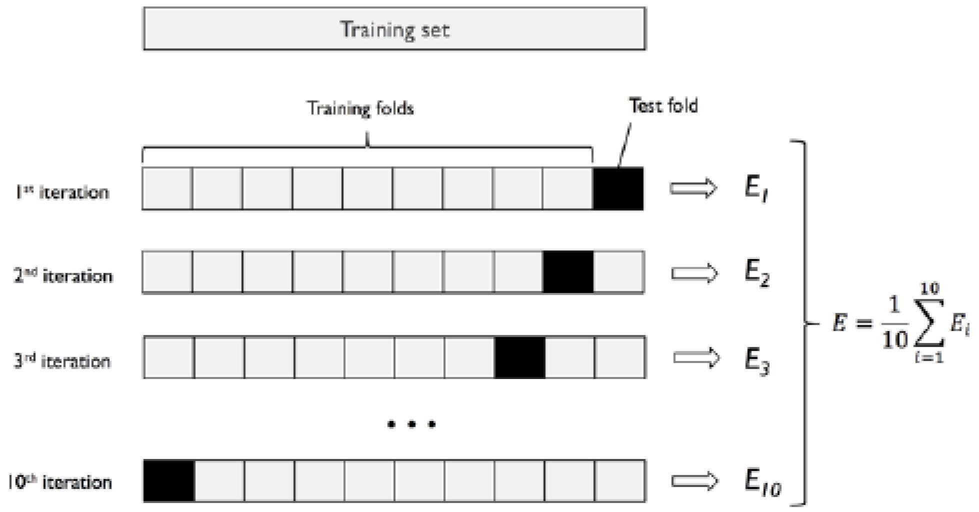


Figure 26. Partitioning the training set for 10-fold cross-validation [54].

K-fold cross-validation provides a useful model performance estimate. However, due to the large number of experiments performed in this study and the large number of hyperparameters tuned for each model, it would drastically increase the computation time in hyperparameter tuning. The only sets of experiments in this thesis where *k*-fold cross-validation was used was when the SVM models' learning curves were computed and when the classification accuracy was calculated for the final models.

Methodology

The flow diagram shown in Figure 27 explains how the model hyperparameters were tuned during the training phase of a model. At first, a speech emotion corpus was selected. Then framing and windowing were applied in data preprocessing. After that, the low-level descriptors were extracted from the audio files, and functionals, such as mean and standard deviations were computed across all audio frames. This resulted in the audio features that were then scaled using standardization. Standardization is performed by subtracting the mean of a feature from that feature value and then dividing it by the feature's standard deviation. This ensures that the feature values have a mean of zero and has a standard deviation of one. Machine learning algorithms like SVM are sensitive to unscaled data. The next step was to partition the data into the training set, the validation set, and the test set. After that, the initial hyperparameter values were set, and the machine learning algorithm was trained on the training set. Once training was done, the model performance was evaluated on validation data. If the performance metrics indicated a case of either overfitting or underfitting, the hyperparameters were re-tuned, and the machine learning algorithm was trained again using the new hyperparameter values. This process was repeated until a fair bias-variance tradeoff was achieved. Once the model was finalized, an unbiased performance evaluation was obtained using the test set. The performance metrics from this final evaluation indicate of how the model will perform when exposed to previously unseen data.

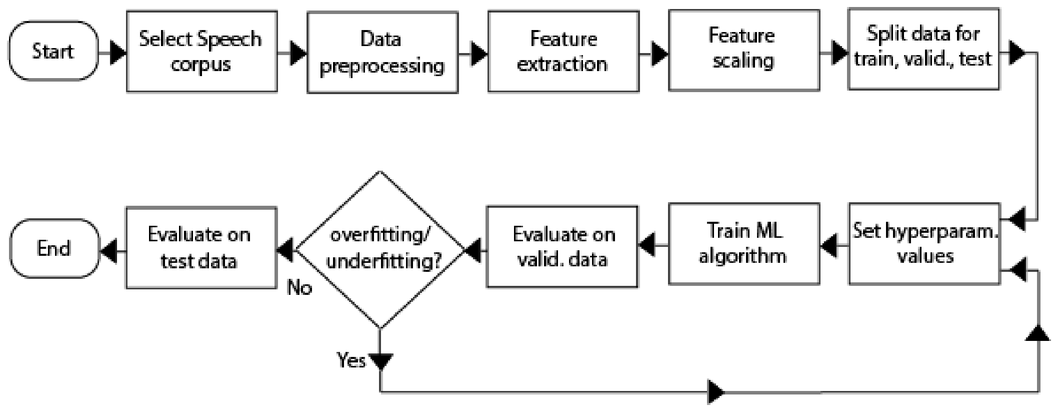


Figure 27. Flow diagram showing the steps of model training and hyperparameter tuning.

The hyperparameter tuning experiments were conducted on the *Complete_Clean_Noise* dataset using only the custom feature set. These hyperparameter values were also used when the algorithms were trained on all other datasets listed in Figure 28. Therefore, after the hyperparameter tuning experiments were completed, 48 experiments were conducted to create the speech emotion recognition system – two feature sets by three machine learning classifiers by eight speech emotion datasets. All these combinations are shown in Figure 28.

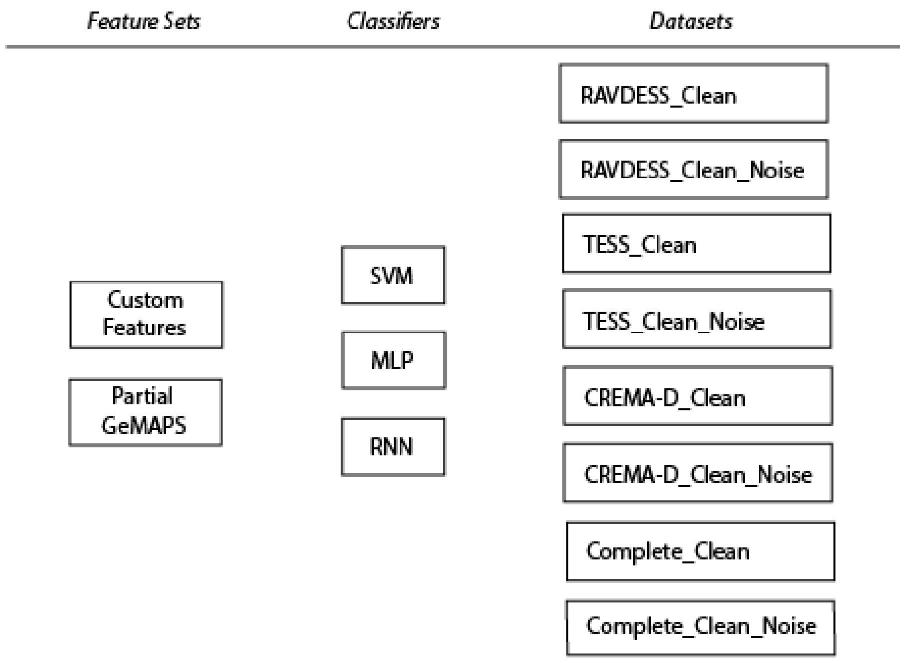


Figure 28. The 48 different experiments conducted for building the speech emotion classifier.

Once all 48 experiments were completed, one model was selected from each type of machine learning classifier. These were combined using *ensemble learning*. In supervised machine learning, ensemble learning is the procedure of combining multiple machine learning algorithms to classify the same data. There are different forms of ensemble learning. The one used in this study is called *voting*. In voting, each machine learning model's predictions are considered, and the final prediction of the system is the class, which occurs the most among the predictions. Once the speech emotion recognition system was finalized using ensemble learning, another round of ensemble learning was performed. This time, the facial expression recognition system created in [45] was combined with the speech emotion recognition system developed in this work to create a robust, multi-modal emotion recognition solution.

Computational Resources

Programming

For this thesis, the Python version 3.7.4 was used for the development of the machine learning models. MATLAB version R2018b was used for background noise addition and verification of results. Table 6 lists all the Python libraries used, along with their versions. The Python projects are susceptible to package versions due to the dependencies among packages. One way to avoid this is to create separate Python environments, i.e., install separate Python versions, for each project.

Table 6. Python libraries used in this research.

Library name	***Version***
scikit-learn	0.23.2
joblib	0.14.1
tensorflow	2.1.0
tensorflow-estimator	2.1.0
h5py	2.10.0
numpy	1.18.5
pandas	1.1.1
matplotlib	3.1.3
praat-parselmouth	0.3.3
librosa	0.8.0
numba	0.48.0

Hardware

Besides personal workstations, Texas State University's LEAP cluster and HiPE research group's servers were used to run the experiments in this study. The majority of the experiments were run on the LEAP cluster. A cluster is a group of servers, and a server is a single workstation computer with high computational capabilities. Each server in LEAP has 28 processing cores, and there are 123 servers available for use. The specifications of the LEAP cluster are given in Table 7.

Table 7. Specifications of the LEAP cluster [60].

CPU type	**Intel Xeon E5-2680v4**
Processor cores	3,532
CPU speed	2.4 GHz
Peak performance	135 TFlops
CPU cores per node	28 (per compute-node)
Nodes	123
Memory	18 TB
Memory per core	4.5 GB (compute)
Disk	48 TB
Operating system	Linux (Cent OS)
Batch system	SLURM

The HiPE servers were used for running some of the experiments. These servers are accessible only to the HiPE research group members and proven to be very useful, mostly when the LEAP cluster was busy. One advantage of the HiPE servers over the LEAP servers is that the HiPE servers have graphics processing units (GPUs) equipped. This additional hardware accelerates model training when deep learning packages such as

TensorFlow are used. TensorFlow is optimized to run on NVIDIA GPUs, making full use of NVIDIA's CUDA cores. Table 8 summarizes the specifications of the HiPE servers.

Table 8. Specifications of the HiPE servers [61].

	PowerEdge C4130 Rack Servers	**PowerEdge R740 Server**
CPU type	Dual Intel Xeon E5-2640 v4 / 2.4GHz / 25M Cache	Dual Intel Xeon Gold / 2.3GHz / 24.75M Cache
CPU cores	20 Cores / 40 Threads	18 Cores / 36 Threads
Memory	16 GB RDIMM x8 Data Width (128GB)	16 GB RDIMM x12 Data Width (192GB)
Disk	Dual 800 GB Solid State Drive uSATA	Dual 1.2TB Solid State Drive SATA
Operating System	Linux (CentOS 7)	Linux (CentOS 7)
Accelerator	NVIDIA Tesla V100 for PCIe (x2)	NVIDIA Tesla V100 for PCIe (x2)

VI. RESULTS AND DISCUSSION

The *Complete_Clean* dataset is comprised of all the original clean speech utterances of the Ryerson Audio-Visual Database of Emotional Speech and Songs (RAVDESS), the Toronto Emotional Speech Set (TESS), and the Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D). The *Complete_Clean_Noise* dataset was created by adding three noise samples in three different SNR values to the *Complete_Clean* dataset. The three noise samples used for background inclusion in the samples are a recording of children playing in a playground, a recording of a shopping mall, and a recording of cars passing by on the streets. SNRs values used are 0 dB, 5 dB, and 10 dB. The noise samples were added to the clean speech utterances using MATLAB, and for each clean speech, a noise sample, an SNR value, and a specific section of the noise file were randomly picked by the MATLAB code. Since the *Complete_Clean_Noise* corpora had class imbalance, the minority classes were resampled (with replacement) to match the sample count of the majority classes. The neutral class samples were lower in both RAVDESS and CREMA-D, and the surprise class was missing from CREMA-D. The hyperparameters of all the models discussed in this section were tuned while being trained on the *Complete_Clean_Noise* dataset. The data split of 80:10:10 was used, where 80% of the dataset was used in training the models, while 10% was used for validation and the other 10% was used for testing. Each data split was *stratified*, meaning that there were equal number of data samples per emotion class in each of the three data splits (training, validation, and test).

Experiments with Custom Feature Set

Python's *Librosa* library was used to extract the custom feature set from all the data samples. The customization included 36 low-level audio descriptors - the Mel-frequency cepstral coefficients (MFCCs), the root-mean-square (RMS) energy, the spectral contrast, and the polynomial coefficients. Among these low-level descriptors, the MFCCs and the RMS energy were used in most of the prior speech emotion recognition related work. The other descriptors were mainly used in music classification tasks. However, they have shown to yield good classification accuracies when applied to emotion classification task in this work. A total of 62 audio features were created using the four low-level audio descriptors of the custom feature set for the SVM and MLP models. They are 26 mean values of first 26 MFCCs across all audio frames, 26 standard deviations of first 26 MFCCs across all audio frames, one mean RMS energy across all audio frames, seven mean values of spectral contrast across all audio frames, and two mean values of polynomial coefficients across all audio frames. No functionals were applied for the RNN model since the low-level descriptors extracted per frame are the *sequences* that the RNN learns from. The low-level descriptors were directly used as the audio features for the RNN.

SVM Model with Custom Feature Set

In Figure 29, the learning curves were plotted for the SVM model trained on the *Complete_Clean_Noise* dataset using the custom feature set. Figure 30 shows the confusion matrix for this model, and Table 9 gives a summary of the results. For this model, the radial basis function (RBF) kernel was used, with C=10.0 and γ=0.01. The *Scikit-learn* library was utilized to develop the SVM model in Python.

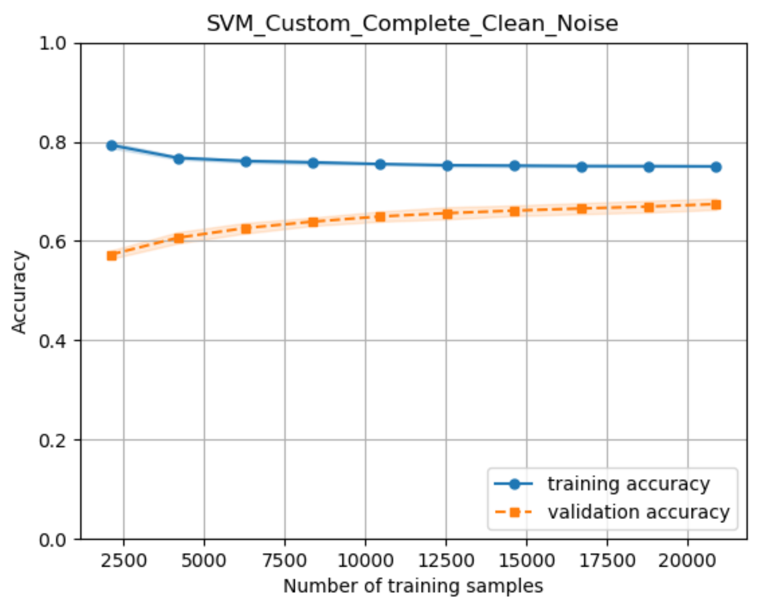
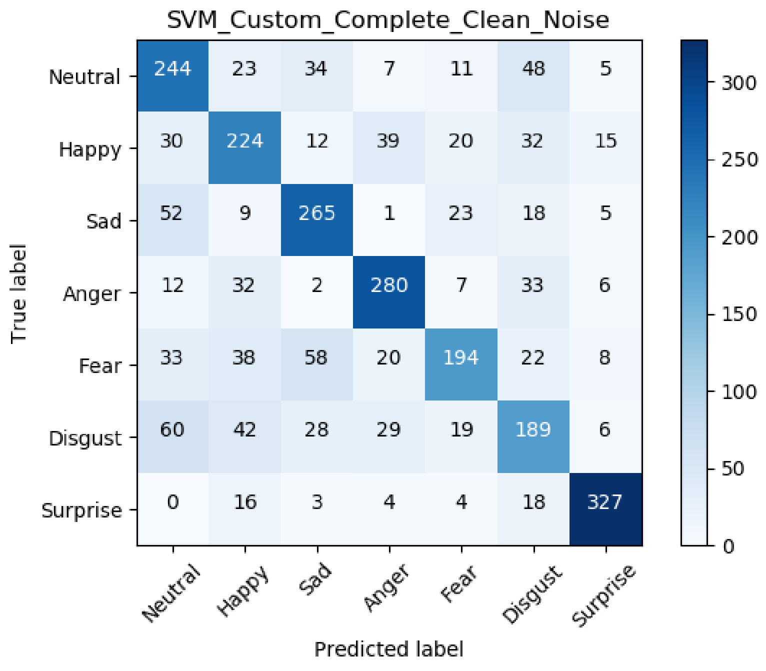


Figure 29. Learning curves for the SVM model trained on the Complete_Clean_Noise corpus, using the custom feature set.



	Neutral	Happy	Sad	Anger	Fear	Disgust	Surprise
Neutral	244	23	34	7	11	48	5
Happy	30	224	12	39	20	32	15
Sad	52	9	265	1	23	18	5
Anger	12	32	2	280	7	33	6
Fear	33	38	58	20	194	22	8
Disgust	60	42	28	29	19	189	6
Surprise	0	16	3	4	4	18	327

Figure 30. Confusion matrix for the SVM model trained on the Complete_Clean_Noise corpus, using the custom feature set.

Table 9. Result summary for the SVM model trained on the Complete_Clean_Noise corpus, using the custom feature set.

Data samples in corpus	26,082 (3,726 samples * 7 emotion classes)
Training : validation : test	80:10:10
Model training time (LEAP)	1 minute and 13 seconds
Training accuracy	75.0 %
Validation accuracy	65.2 %
Test accuracy	66.1 %
Precision	66.4 %
Recall	66.1 %

The learning curves of this model show that the validation accuracy closely follows the training accuracy. This is due to the values picked for the C and γ parameters, which ensured that the model did not overfit the training data. In the confusion matrix of the experiment, if all numbers across a row are added, it gives the total number of data samples in the test set for the emotion label mentioned in that row's name. Since the test set was equal to 10 % of the entire dataset, it contained a total of 2,604 data samples. For the stratified test set with seven emotion classes, this resulted in 2,604/7 = 372 data samples per emotion class, each is equal to the sum of the numbers in each row of the confusion matrix. From the confusion matrix, it can be seen that the surprise emotion was the most accurately detected emotion, followed by anger, sadness, and neutral. The time taken to train the SVM model on the LEAP server was only 1 minute and 13 seconds.

MLP Model with Custom Feature Set

The number of artificial neuron units used in an artificial neural network and the number of layers are hyperparameters that can be tuned for getting high accuracies. There is no golden rule for selecting the number of neurons or layers. Researchers usually experiment with these parameters and select values that provide the highest performance. A common convention among computer scientists is to use *log* BASE-2 number, like 64, 128, and 256 [62]. Another convention is to use increments of 50 or hundred, like 50, 100, and 200 [63]. The number of input-layer neurons is equal to the number of input features, and the number of output-layer neurons is equal to the number of classes in the dataset. Even though there is no rule for selecting the number of neurons in the hidden layer(s), there are some rules-of-thumb that can be followed, according to [64]. To design the MLP architecture of this model, number of neurons, such as 10, 50, 100, 200, and 500, were selected for each layer. The rules-of-thumb described in [64] were used to select the final number of neurons and layers for the high-performing architectures. The architecture for the MLP model that used the custom feature set is shown in Figure 31. There are 62 units in the input layer, which correspond to the number of input features in the custom feature set. The first hidden layer has 105 units, which is 170% of the number of input units used. The second hidden layer has 62 units, which is equal to the number of input neurons. Finally, the output layer has seven units, corresponding to the seven emotion classes used in this work.

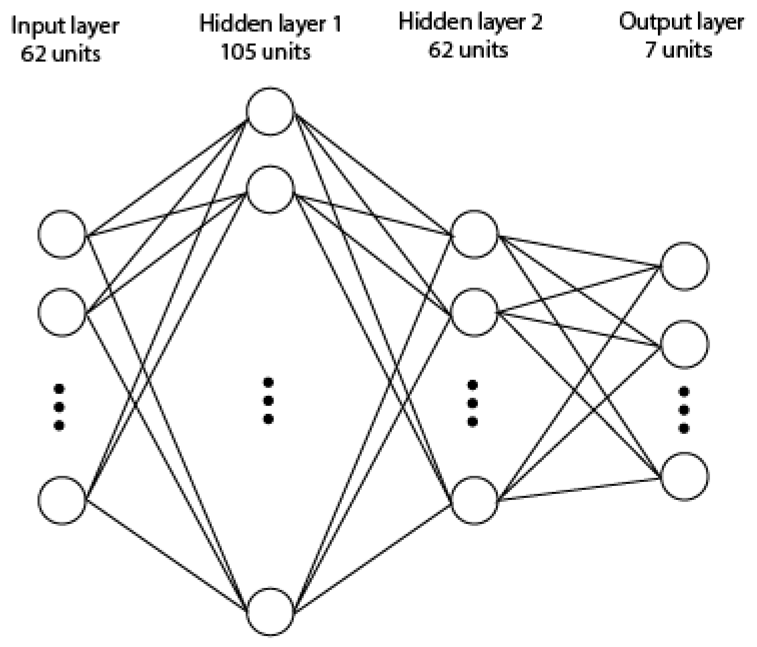


Figure 31. Architecture of the MLP used with the custom feature set.

The Adam optimizer was used in order to minimize the loss function, which in this case is the categorical cross-entropy loss for the MLP model. The rectified linear unit (ReLU) activation function was used and for the output units, the Softmax activation function was used, which provides the prediction accuracies for each class for the hidden layer units. Instead of using a fixed learning rate, a learning rate scheduler was used to change the learning rate as the training progressed. An inverse time decay function was used as the learning rate schedule, with an initial learning rate of 0.01, 1000 decay steps, and a decay rate of 80%. The training, validation, and testing data were each divided into batches of size sixteen, and 50 epochs were used during training. *Dropout* is a regularization technique where a fraction of the connections between the hidden layer neurons are randomly dropped during training. This ensures that the model is not overfitting to the training data. During the validation and testing phases, however, all the neurons are connected, i.e., no dropout is used. After trying out different dropouts at different positions, the best combination was used. A dropout of 30% was used between the two hidden layers, and a 10 % dropout was used between the second hidden layer and the output layer. The accuracy curves for the MLP model are plotted in Figure 32 while the loss curves are plotted in Figure 33. Figure 34 shows the confusion matrix of this model. The result summary is given in Table 10. The model was created using the *Keras* API from the *TensorFlow* library in Python.

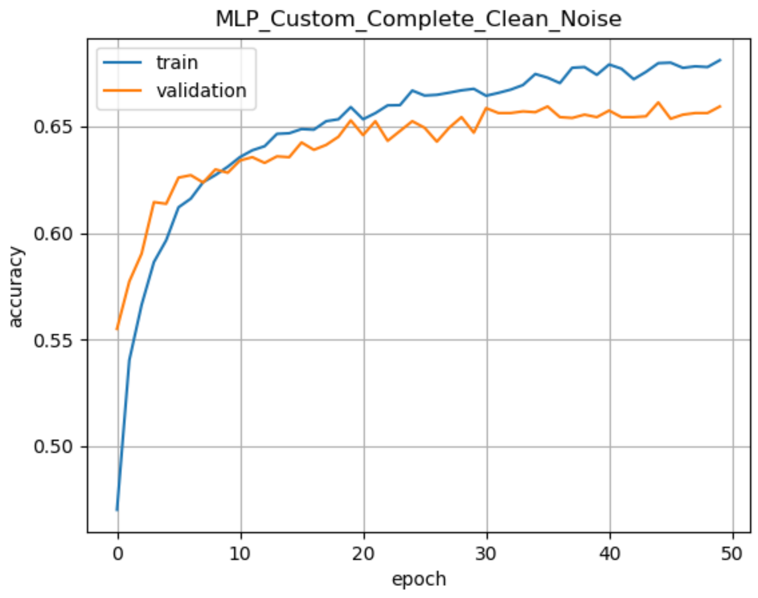


Figure 32. Accuracy curves for the MLP model trained on the Complete_Clean_Noise corpus, using the custom feature set.

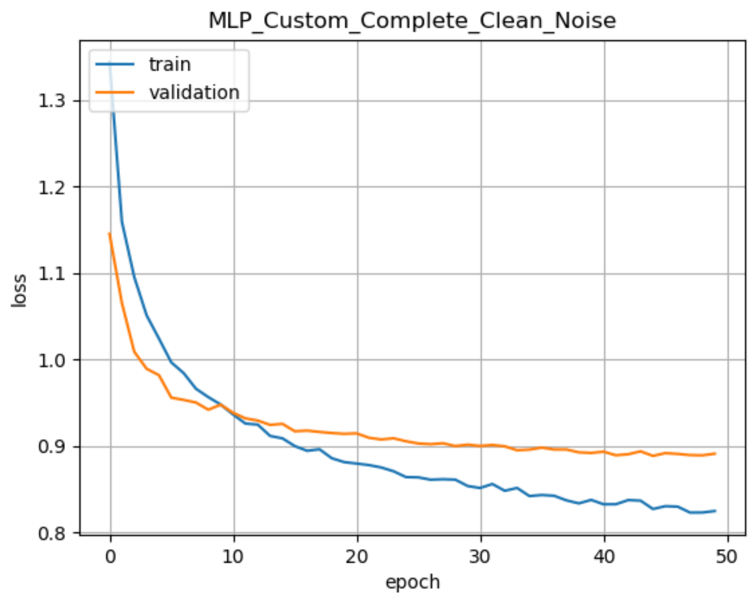
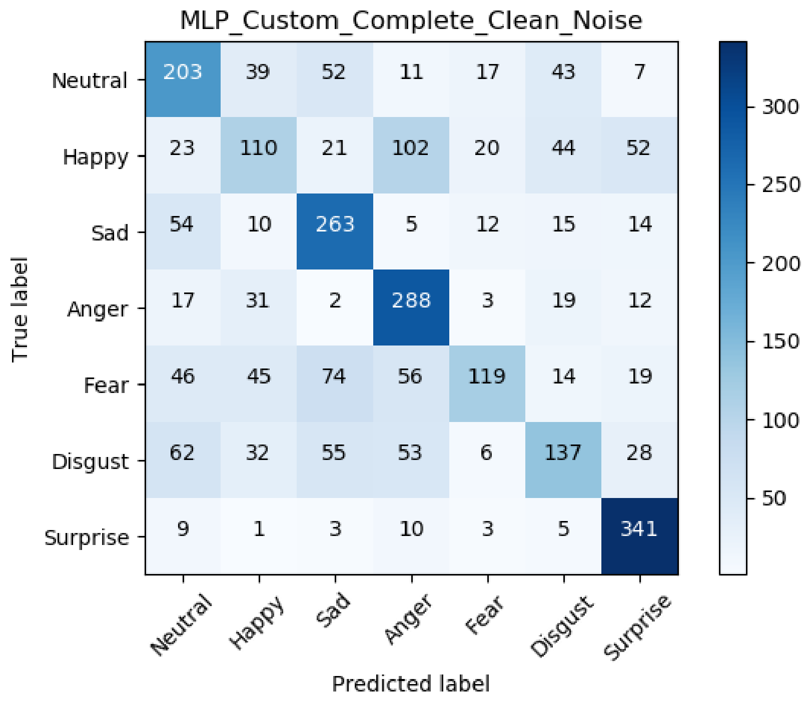


Figure 33. Loss curves for the MLP model trained on the Complete_Clean_Noise corpus, using the custom feature set.



	Neutral	Happy	Sad	Anger	Fear	Disgust	Surprise
Neutral	203	39	52	11	17	43	7
Happy	23	110	21	102	20	44	52
Sad	54	10	263	5	12	15	14
Anger	17	31	2	288	3	19	12
Fear	46	45	74	56	119	14	19
Disgust	62	32	55	53	6	137	28
Surprise	9	1	3	10	3	5	341

Figure 34. Confusion matrix for the MLP model trained on the Complete_Clean_Noise corpus, using the custom feature set.

Table 10. Result summary for the MLP model trained on the Complete_Clean_Noise corpus, using the custom feature set.

Data samples in corpus	26,082 (3,726 samples * 7 emotion classes)
Training : validation : test	80:10:10
Model training time (LEAP)	1 minute and 29 seconds
Training accuracy	68.1 %
Validation accuracy	65.9 %
Test accuracy	65.7 %
Precision	83.3 %
Recall	50.2 %

For this MLP model, the accuracy curves are very close, and the same can be observed for the loss curves. This is a sign of a properly tuned neural network. During the first round of experiments, the model was trained without any regularization. The resulting curves and classification scores indicated a huge overfitting issue. After introducing dropout between the layers, the gap between the validation accuracy and the training accuracy was reduced. Also, the learning rate was kept constant during the first few experiments. The problem with that was the optimizer kept overshooting the minimum loss due to the fixed learning rate being unnecessarily high at that stage of the training. This was visible from the rising loss curves. After using a learning rate scheduler, which slowly reduced the learning rate as training progressed, the losses seemed to decrease consistently. The test accuracy of this model was about one percent less than that of the SVM model. However, the precision score was higher in this model. The training of this MLP model was halted after 50 epochs because the validation loss was almost steady after the 50th epoch. Again, the surprise emotion was the most accurately classified emotion, followed by anger and sadness, looking at the confusion matrix. The training time of this model was similar to that of the SVM.

RNN Model with Custom Feature Set

The RNN layers created using *Keras* requires a tensor as the input, compared to the 2D-structured inputs in MLP (*batch, features*). A tensor is a three-dimensional array of numbers. For RNNs, the three dimensions are the number of data samples, the number of features, and the number of time steps (*batch, time steps, features*). The audio frames were used as the time steps, while the features were the low-level descriptors extracted per frame to process the sequential data. For this reason, all the low-level descriptors extracted using the custom feature set were used as the audio features. In this case, each low-level descriptor value is extracted for each audio frame, which forms a sequence of data suitable to be processed by an RNN. The audio frames represent the time steps of the input data. Meaning, once the current audio frame has been processed, with all the low-level descriptors extracted and fed to the network, the next audio frame is processed. Using a sampling rate of 16 kHz and a frame length of 512 samples (32 ms), around 150 audio frames were processed per audio file. The technique used to determine the number of neurons in each layer was similar to the one used in the MLP model trained on the *Complete_Clean_Noise* dataset using the custom feature set. However, since LSTM layers are used instead of standard RNN layers, each hidden recurrent unit is an LSTM cell. Figure 7 shows the architecture for the RNN network used in this model. The input layer has 36 units, corresponding to the 36 low-level descriptors of the custom feature set.

The first LSTM has 36 cells, and the second LSTM layer has 12 cells. The output layer has seven units. Figure 35 shows the architecture of the RNN model. The LSTM cells are represented by a *recurrent edge* on the units.

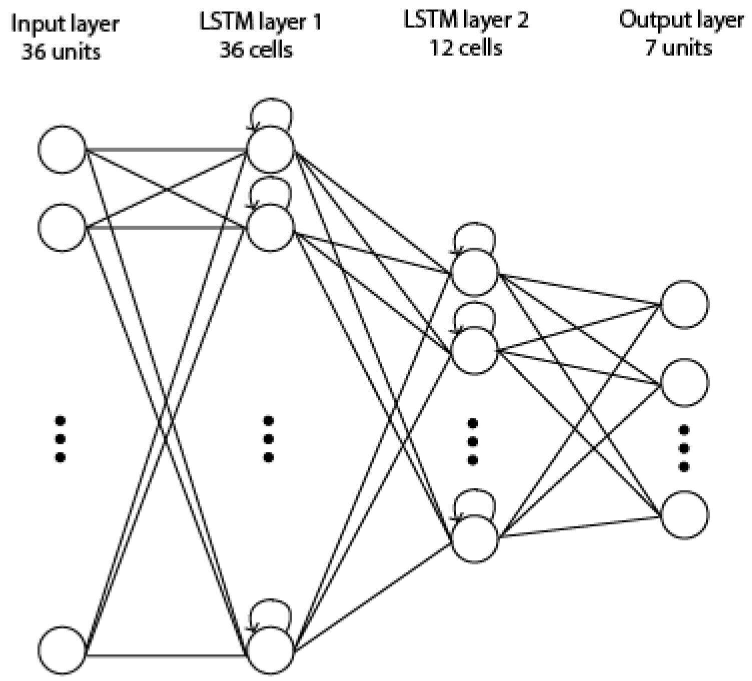


Figure 35. Architecture of the RNN used with the custom feature set.

The Adam optimizer was used along with the categorical cross-entropy loss function. The hyperbolic tangent (tanh) function was the activation function for the LSTM cells, and the sigmoid (σ) function was the recurrent activation function. The Softmax function was used as the activation for the output units. The inverse time decay function is the learning rate scheduler, with 0.01 as the initial learning rate, 1,000 as the decay steps, and 80% as the decay rate. A batch size of sixteen was used, and the total number of epochs used during training was 50. A 30% dropout was placed between the two LSTM layers and a 30% dropout between the second LSTM layer and the output. A 20% recurrent dropout was placed for the LSTM cells in the first layer. Figure 36 shows the accuracy curves of the RNN model, and Figure 37 shows the loss curves. The confusion matrix for this model is shown in Figure 38. Table 11 gives the result summary.

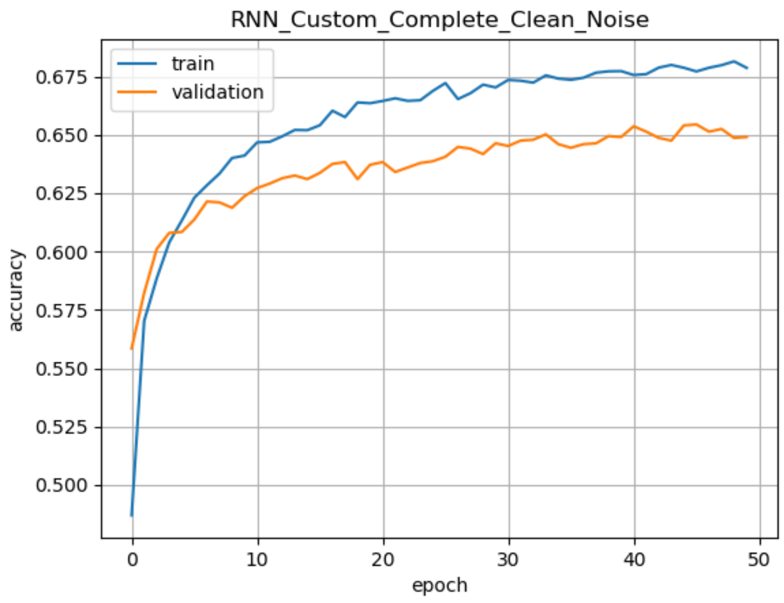


Figure 36. Accuracy curves for the RNN model trained on the Complete_Clean_Noise corpus, using the custom feature set.

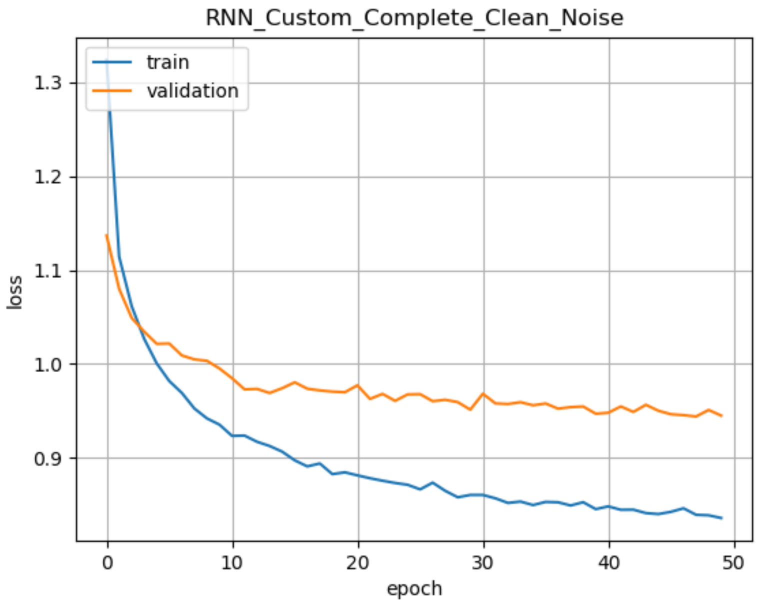


Figure 37. Loss curves for the RNN model trained on the Complete_Clean_Noise corpus, using the custom feature set.

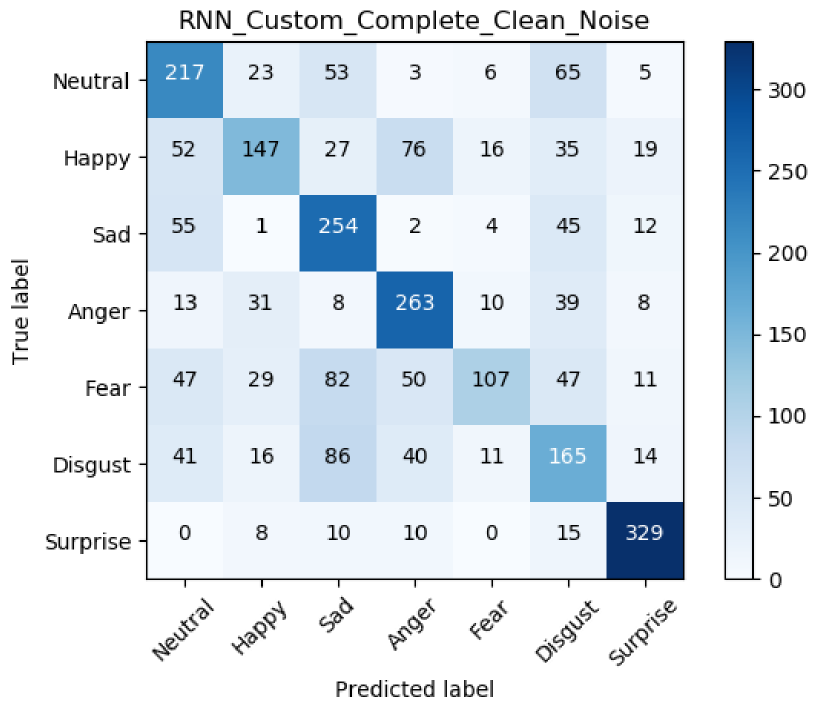


Figure 38. Confusion matrix for the RNN model trained on the Complete_Clean_Noise corpus, using the custom feature set.

Table 11. Result summary for the RNN model trained on the Complete_Clean_Noise corpus, using the custom feature set.

Data samples in corpus	26,082 (3,726 samples * 7 emotion classes)
Training : validation : test	80:10:10
Model training time (LEAP)	46 minute and 37 seconds
Training accuracy	67.9 %
Validation accuracy	64.9 %
Test accuracy	63.7 %
Precision	75.3 %
Recall	53.7 %

Just as it was for the MLP model, the validation loss for the RNN model stopped decreasing after the 50th epoch, which is why the training was stopped at that point. Using more epochs would overexpose the model to the training data, resulting in a higher training accuracy but lower generalizing ability to previously unseen data. The RNN model took over 46 minutes to train, which is the longest training time among all three models. The average precision of this model was higher than that for the SVM model. Compared to the MLP model, the classification accuracies and precision score were slightly lower in this model – the RNN model had a higher average recall. Like the SVM model, the top four most accurately predicted emotions for this model were surprise, anger, sad, and neutral.

Experiments with Partial GeMAPS Feature Set

The Geneva Minimalist Acoustic Parameter Set (GeMAPS) contains a total of 62 audio features. An extended version of the feature set, extended GeMAPS (eGeMAPS), contains 88 audio features. The GeMAPS features can be extracted using the *OpenSMILE* toolkit. The use of any external toolkits was avoided for feature extraction since this study aimed to build a speech emotion recognition system using Python. Even if it is possible to use both OpenSMILE and Python to predict real-time emotions, the cross-platform implementation would likely increase the computation time. Just like OpenSMILE, *PRAAT* is another useful software package that allows users to perform various types of audio processing. The *Parselmouth* library in Python allows the users to use PRAAT's functionality in Python scripts directly. This library was used to extract the features from this feature set. Since it was not possible to extract all the features of GeMAPS using *Parselmouth*, a total of 36 audio features were selected from the eGeMAPS feature set and used in this work – hence the reason for calling it the "partial" GeMAPS feature set. These 36 features were obtained using functionals on fifteen low-level descriptors – pitch (log F_0), loudness (intensity), the first three formants frequencies (F_1, F_2, and F_3), the bandwidths of the first three formants, the first four MFCCs, harmonics-to-noise ratio (HNR), jitter, and shimmer. For the SVM and MLP models, 36 features were used to train the algorithms. These 36 features include the mean, the standard deviation, the 20th percentile, the 50th percentile, the 80th percentile, and the range of 20th to 80th percentile of the pitch and loudness, and the mean and standard deviations for the MFCCs, formant frequencies, formant bandwidths, and HNR, and local absolute jitter, and local shimmer. For the RNN model, the fifteen low-level descriptors, extracted per audio frame, were used as the audio features.

SVM Model with Partial GeMAPS Feature Set

In Figure 39, the learning curves were plotted for the support vector machine (SVM) model trained on the *Complete_Clean_Noise* dataset using the partial GeMAPS set. Figure 40 shows the confusion matrix for this model, and Table 12 summarizes the results. This model, just like the SVM model created using the custom feature set, uses the RBF kernel, with C=10.0 and γ=0.01. Using these hyperparameter values yielded the best bias-variance tradeoff out of all values used.

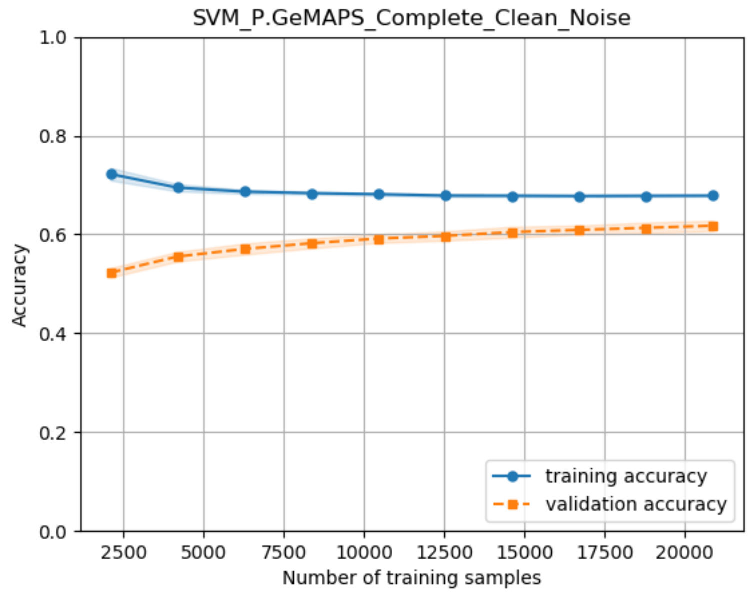


Figure 39. Learning curves for the SVM model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

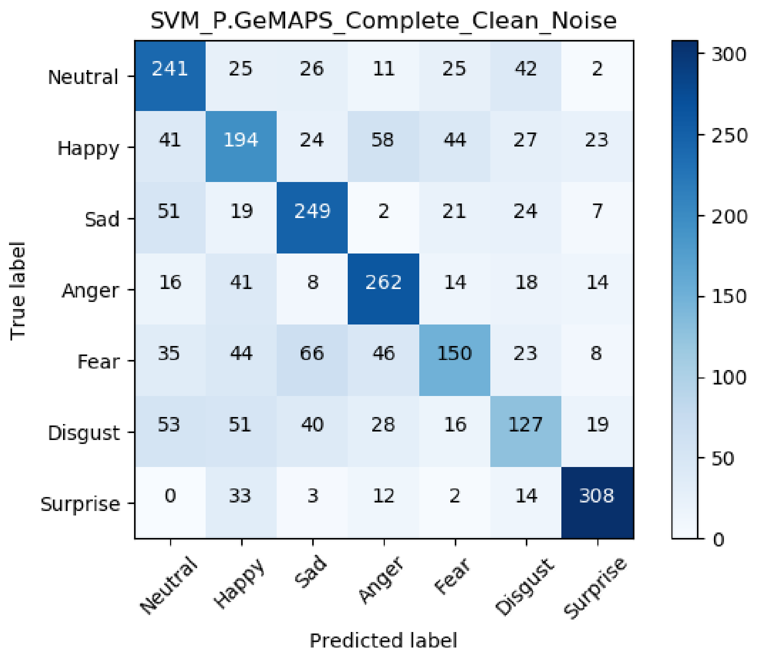


Figure 40. Confusion matrix for the SVM model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

Table 12. Result summary for the SVM model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

Data samples in corpus	26,082 (3,726 samples * 7 emotion classes)
Training : validation : test	80:10:10
Model training time (LEAP)	55 seconds
Training accuracy	68.0 %
Validation accuracy	58.7 %
Test accuracy	58.7 %
Precision	58.2 %
Recall	58.7 %

The SVM model trained using the partial GeMAPS feature set showed an overall lower performance than the SVM model trained using the custom feature set. However, despite being around half the custom feature set's size, the differences in the two feature set models' performance metrics were only about ten percent. This validates the usefulness of the partial GeMAPS features in speech emotion recognition. The lower training time is due to the lower number of features used during training. Furthermore, just like the previous SVM model, this model could predict the surprise emotion samples with the highest accuracy, followed by anger, sad, and neutral. The learning curves show no signs of overfitting.

MLP Model with Partial GeMAPS Feature Set

For the MLP model, the network's architecture was constructed similarly to the one constructed for the MLP model that used the custom feature set. The network was scaled down to account for the lower number of input features. The input layer had 36 units. The first hidden layer had 61 units, and the second hidden layer had 36 units. The output layer had seven units. All other hyperparameter values were kept the same from the MLP model that used the custom feature set, as this gave the best bias-variance tradeoff. The accuracy curves for this MLP model are plotted in Figure 41, while the loss curves are plotted in Figure 42. Figure 43 shows the confusion matrix of this model. The result summary is given in Table 13. It can be noticed that the validation loss is lower than the training loss. This can be attributed to the use of dropout during training.

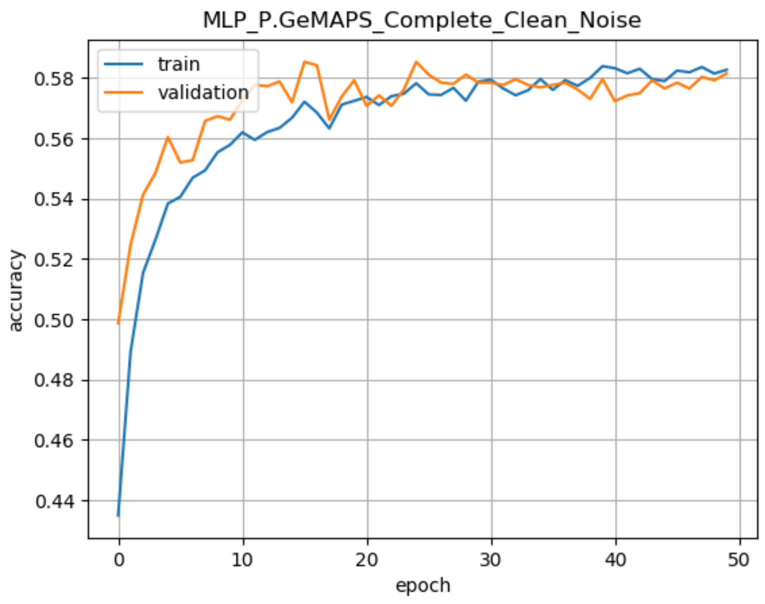


Figure 41. Accuracy curves for the MLP model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

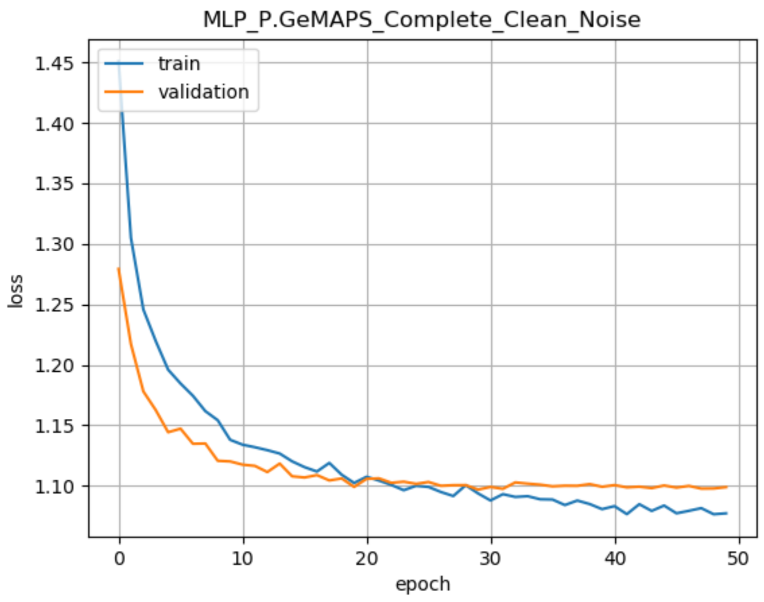


Figure 42. Loss curves for the MLP model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

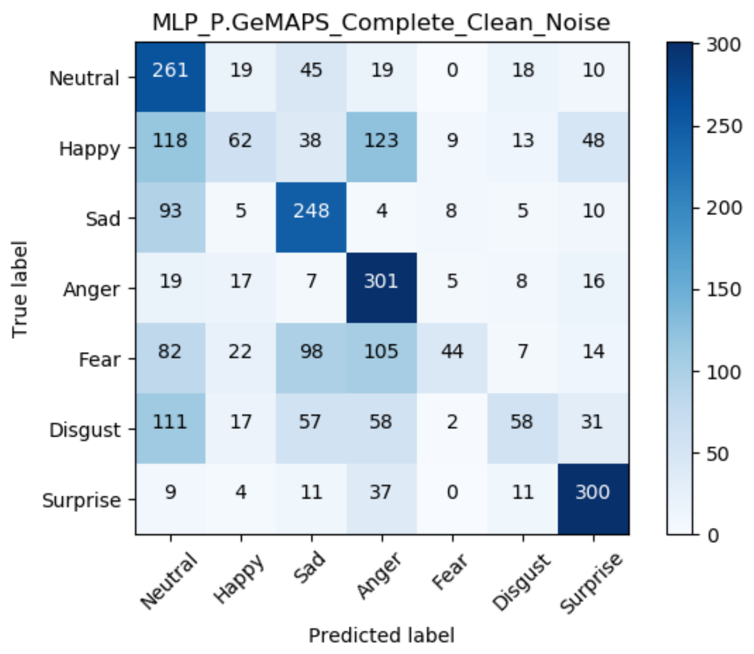


Figure 43. Confusion matrix for the MLP model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

Table 13. Result summary for the MLP model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

Data samples in corpus	26,082 (3,726 samples * 7 emotion classes)
Training : validation : test	80:10:10
Model training time (LEAP)	1 minute and 23 seconds
Training accuracy	58.3 %
Validation accuracy	58.2 %
Test accuracy	57.9 %
Precision	79.5 %
Recall	38.2 %

Comparing this model with the SVM model created using the partial GeMAPS feature set, it can be seen that the model performs better overall. The top four correctly classified emotions remain to be surprise, anger, sad, and neutral. However, the anger class was more correctly classified than the surprise class, and the neutral class was more correctly classified than the sad class. For this model, the accuracy curves are very similar, indicating a well-tuned model, although most of the hyperparameter values were imported from the MLP model trained using the custom feature set. Moreover, even though the performance scores of this MLP model were lower than the performance scores of the MLP model trained using the custom feature set, the average precision value of this model is very close to the other MLP model. The MLP model trained using the partial GeMAPS could predict more neutral class samples than the MLP model trained using the custom feature set.

RNN Model with Partial GeMAPS Feature Set

The RNN model developed using partial GeMAPS features on the *Complete_Clean_Noise* had fifteen input neurons, corresponding to the fifteen low-level descriptors extracted per audio frame (time step). The first LSTM layer had fifteen cells, and the second LSTM layer had twelve cells. The output layer had seven units. The dropouts were reduced from the previous RNN model to 20 %. All other hyperparameter values were imported from the RNN model that used the custom feature set. The accuracy curves for this RNN model are shown in Figure 44, and the loss curves are shown in Figure 45. Figure 46 shows the confusion matrix, and Table 14 shows list the result summary.

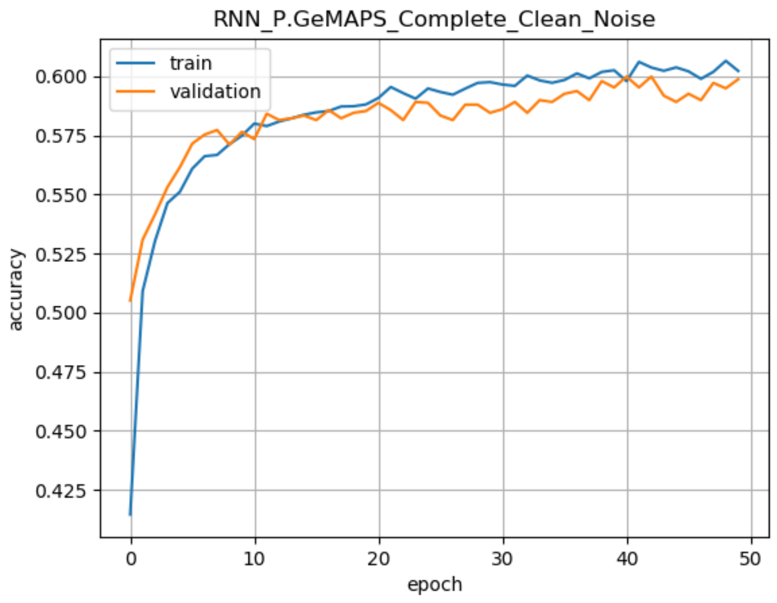


Figure 44. Accuracy curves for the RNN model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

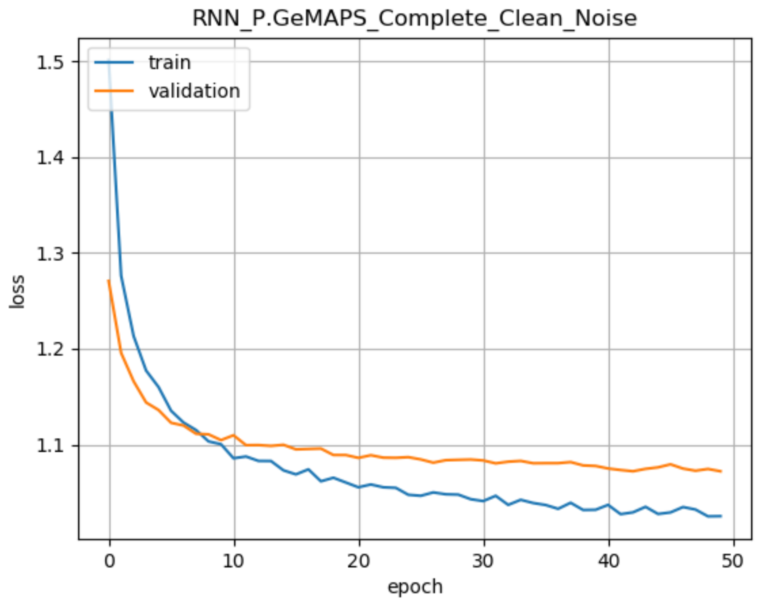


Figure 45. Loss curves for the RNN model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

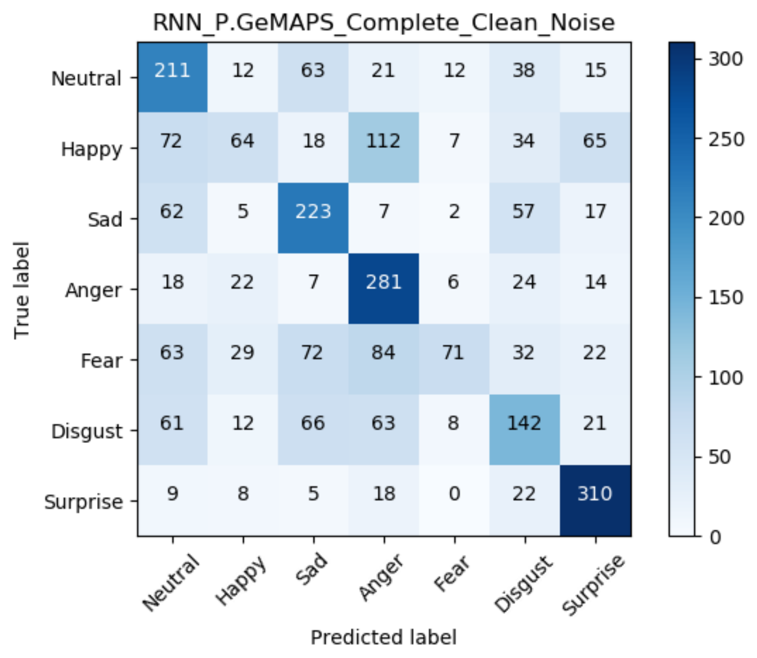


Figure 46. Confusion matrix for the RNN model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

Table 14. Result summary for the RNN model trained on the Complete_Clean_Noise corpus, using the partial GeMAPS feature set.

Data samples in corpus	26,082 (3,726 samples * 7 emotion classes)
Training : validation : test	80:10:10
Model training time (LEAP)	16 minutes
Training accuracy	60.2 %
Validation accuracy	59.9 %
Test accuracy	57.9 %
Precision	75.7 %
Recall	41.9 %

Two interesting occurrences can be seen for the performance measures of this RNN model. The first is that this model had a higher average precision score than the RNN model trained using the custom feature set. Secondly, the validation and test accuracies are only 5 % lower than for the RNN model trained using the custom feature set.

Comparing Custom Feature Set Models

The SVM model created using the custom feature set was tested on the other datasets used in this work. The results are given in Table 15. Similarly, the performances of the MLP and RNN models using a custom feature set was also compared in Tables 16 and 17, respectively. It can be observed from these tables that the MLP models had the highest precision score, while the SVM models showed the lowest precision score. The SVM, MLP, and RNN models performed extraordinarily well with the TESS dataset. The TESS dataset had only two female participants who recorded 200 similar-sounding sentences in all seven emotions, with no intensity change. This lack of variation did not pose the models quickly picked up any problems for the models during prediction as to the data samples' features. All models had the lowest performance metrics for the CREMA-D dataset. This speech emotion corpus had the highest number of participants - a total of 91 actors. Furthermore, each recording was done in four different emotional intensities. The models struggled to learn all the variations in the data.

Also, some of the models were overfitting to the training data. This is mainly because the hyperparameters were tuned on the *Complete_Clean_Noise* dataset and were not specifically tuned to other datasets. Tuning the hyperparameters for each speech corpus would make the comparison between the models invalid.

Table 15. Comparing the performance of the SVM model created using the custom feature set on different datasets.

Corpus	*Training %*	*Valid. %*	*Test %*	*Precision*	*Recall*
RAVDESS_Clean	99.0 %	85.8 %	80.6 %	81.8 %	80.6 %
RAVDESS_Clean_Noise	87.0 %	61.6 %	58.2 %	58.9 %	58.2 %
TESS_Clean	100 %	99.6%	100 %	100 %	100 %
TESS_Clean_Noise	99.0 %	98.6 %	97.7 %	97.7 %	97.7 %
CREMA-D_Clean	86.0 %	54.9 %	57.6 %	57.7 %	57.6 %
CREMA-D_Clean_Noise	79.0 %	51.1 %	52.1 %	51.9 %	52.1 %
Complete_Clean	81.0 %	71.9 %	73.0 %	74.3 %	73.0 %
Complete_Clean_Noise	75.0 %	65.2 %	66.1 %	66.4 %	66.1 %

Table 16. Comparing the performance of the MLP model created using the custom feature set on different datasets.

Corpus	*Training %*	*Valid. %*	*Test %*	*Precision*	*Recall*
RAVDESS_Clean	94.5 %	82.8 %	77.6 %	77.9 %	76.1 %
RAVDESS_Clean_Noise	80.6 %	64.2 %	56.7 %	66.3 %	50.0 %
TESS_Clean	99.9 %	100.0 %	100.0 %	100.0 %	100.0 %
TESS_Clean_Noise	99.5 %	98.0 %	98.8 %	98.8 %	98.8 %
CREMA-D_Clean	72.7 %	54.5 %	54.7 %	61.5 %	44.5 %
CREMA-D_Clean_Noise	60.7 %	51.9 %	54.5 %	65.7 %	36.2 %
Complete_Clean	75.9 %	70.8 %	69.5 %	80.5 %	60.0 %
Complete_Clean_Noise	68.1 %	65.9 %	65.7 %	83.3 %	50.3 %

Table 17. Comparing the performance of the RNN model created using the custom feature set on different datasets.

Corpus	*Training %*	*Valid. %*	*Test %*	*Precision*	*Recall*
RAVDESS_Clean	94.6 %	76.1 %	61.2 %	61.2 %	59.0 %
RAVDESS_Clean_Noise	85.0 %	59.3 %	51.5 %	55.9 %	49.6 %
TESS_Clean	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
TESS_Clean_Noise	99.8 %	99.3 %	99.6 %	99.6 %	99.6 %
CREMA-D_Clean	67.7 %	49.0 %	55.0 %	58.3 %	47.1 %
CREMA-D_Clean_Noise	59.9 %	51.2 %	51.0 %	58.7 %	40.5 %
Complete_Clean	74.3 %	66.5 %	64.5 %	72.1 %	57.3 %
Complete_Clean_Noise	67.9 %	64.9 %	63.7 %	75.4 %	53.7 %

Comparing Partial GeMAPS Feature Set Models

The performances of the SVM, MLP, and RNN models created using the partial GeMAPS feature set were compared when used with other datasets used in this work. The results are given in Tables 18, 19, and 20, respectively. The results obtained from these models were similar to those obtained from the models trained using the custom feature set, except that the classification accuracies were lower. The SVM models had the worst precision scores, and the MLP models had the highest precision scores. The SVM, MLP, and RNN models performed best on the TESS corpus and struggled the most on the CREMA-D corpus.

Table 18. Comparing the performance of the SVM model created using the partial GeMAPS feature set on different datasets.

Corpus	*Training %*	*Valid. %*	*Test %*	*Precision*	*Recall*
RAVDESS_Clean	90.0 %	66.4 %	56.7 %	57.7 %	56.7 %
RAVDESS_Clean_Noise	76.0 %	52.2 %	52.6 %	54.3 %	52.6 %
TESS_Clean	99.0 %	97.1 %	99.3 %	99.3 %	99.3 %
TESS_Clean_Noise	97.0 %	91.4 %	94.5 %	94.5 %	94.5 %
CREMA-D_Clean	69.0 %	52.1 %	54.2 %	53.4 %	54.2 %
CREMA-D_Clean_Noise	61.0 %	50.5 %	49.6 %	49.4 %	49.6 %
Complete_Clean	76.0 %	62.7 %	65.2 %	65.9 %	65.2 %
Complete_Clean_Noise	68.0 %	58.7 %	58.7 %	58.2 %	58.7 %

Table 19. Comparing the performance of the MLP model created using the partial GeMAPS feature set on different datasets.

Corpus	*Training %*	*Valid. %*	*Test %*	*Precision*	*Recall*
RAVDESS_Clean	80.7 %	68.7 %	55.2 %	59.8 %	50.0 %
RAVDESS_Clean_Noise	64.9 %	52.2 %	52.6 %	65.8 %	50.0 %
TESS_Clean	99.5 %	98.6 %	98.6 %	98.6 %	98.6 %
TESS_Clean_Noise	94.8 %	92.9 %	94.5 %	94.8 %	93.9 %
CREMA-D_Clean	59.6 %	53.0 %	54.9 %	64.4 %	36.4 %
CREMA-D_Clean_Noise	52.1 %	49.3 %	49.1 %	61.3 %	25.8 %
Complete_Clean	67.6 %	62.3 %	62.2 %	79.9 %	48.1 %
Complete_Clean_Noise	58.3 %	58.2 %	57.9 %	79.5%	38.2 %

Table 20. Comparing the performance of the RNN model created using the partial GeMAPS feature set on different datasets.

Corpus	*Training %*	*Valid. %*	*Test %*	*Precision*	*Recall*
RAVDESS_Clean	86.5 %	50.8 %	47.0 %	50.0 %	43.3 %
RAVDESS_Clean_Noise	72.6 %	50.0 %	39.2 %	44.7 %	34.3 %
TESS_Clean	99.7 %	98.2 %	99.3 %	99.3 %	99.3 %
TESS_Clean_Noise	99.8 %	98.4 %	98.8 %	98.8 %	98.8 %
CREMA-D_Clean	61.1 %	48.2 %	45.7 %	52.4 %	36.4 %
CREMA-D_Clean_Noise	52.9 %	45.3 %	46.0 %	55.2 %	26.8 %
Complete_Clean	67.3 %	61.3 %	59.3 %	70.0 %	48.0 %
Complete_Clean_Noise	60.2 %	59.9 %	57.9 %	75.7 %	41.9 %

Comparing Models from Both Feature Sets

Table 21 shows the performance metrics for the models created using the custom feature set and the models created using the partial GeMAPS feature set. The models listed in this table were trained and evaluated on the *Complete_Clean_Noise* dataset. From Table 21, it can be seen that the custom feature set models have outperformed the partial GeMAPS feature set in all metrics. This can be attributed to the significantly lower number of features in the partial GeMAPS feature set since the lower number of features could not capture the variations in the training data. The model that showed the best performance among all the models studied in this work was the MLP model trained on the *Complete_Clean_Noise* dataset using the custom feature set. It showed the highest classification accuracies and good average precision and average recall scores as well. Figure 47 shows the results of stratified *10-fold cross-validation* for the models on the *Complete_Clean_Noise* dataset. After separating the test set from the training set, the training set was split into ten equal parts or folds. The model evaluation was performed ten times, and each time one out of the ten parts was used as the validation set while the remaining nine parts were used for the test set. Each time, a different fold was selected for validation split, and the training and validation accuracy were calculated for the ten experiments. The accuracy scores shown in the bar plot of Figure 47 were calculated by computing the mean of all the classification accuracies over the ten experiments.

Table 21. Comparing the models created using the two different feature sets, for the Complete_Clean_Noise corpus.

Classifier	*Feature Set*	*Training %*	*Valid. %*	*Test %*	*Precision*	*Recall*
SVM	Custom	75.0 %	65.2 %	66.1 %	66.4 %	66.1 %
SVM	P.GeMAPS	68.0 %	58.7 %	58.7 %	58.2 %	58.7 %
MLP	Custom	68.1 %	65.9 %	65.7 %	83.3 %	50.3 %
MLP	P.GeMAPS	58.3 %	58.2 %	57.9 %	79.5%	38.2 %
RNN	Custom	67.9 %	64.9 %	63.7 %	75.4 %	53.7 %
RNN	P.GeMAPS	60.2 %	59.9 %	57.9 %	75.7 %	41.9 %

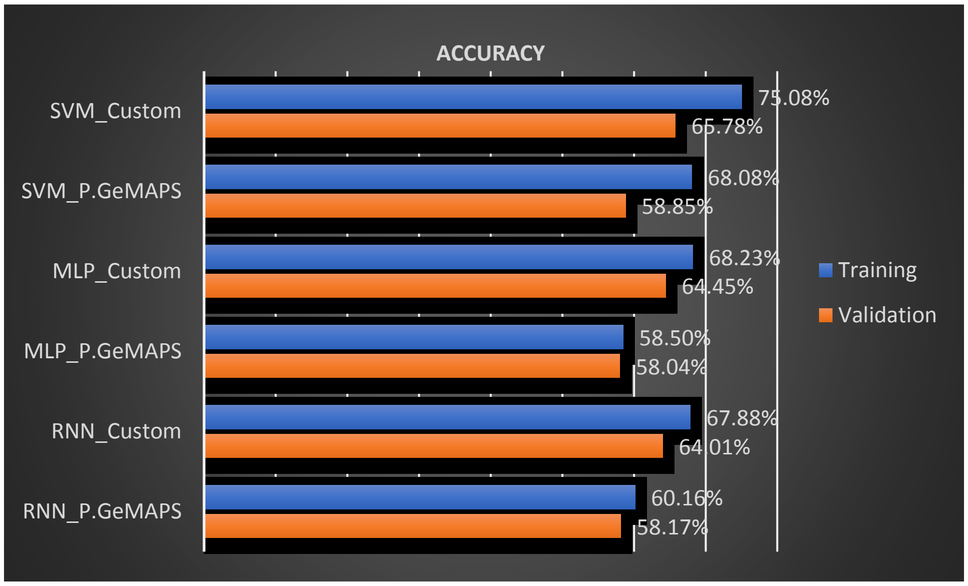


Figure 47. 10-fold cross-validation results on the Complete_Clean_Noise corpus for the models listed on the vertical axis.

When comparing the confusion matrices for the models trained on the *Complete_Clean_Noise* dataset, it can be seen that the surprise emotion was the class that was most accurately predicted. This can be due to the fact that the surprise class was missing from the CREMA-D dataset, and when the three datasets were joined to create the *Complete_Clean_Noise* dataset, this class was heavily resampled from RAVDESS and TESS. Also, the neutral class samples were lower in the RAVDESS and CREMA-D datasets, so it was also resampled when all three datasets were joined. However, resampling was done *after* the training, validation, and test samples were separated, which prevented the repetition of minority data samples in the three splits. Besides surprise and neutral emotions, the top two most accurately classified emotions were sad emotions and anger. For the RAVDESS dataset, the surprise emotion was the most accurately predicted class. For the CREMA-D dataset, the anger emotion was the most accurately predicted class. For TESS, the neutral emotion was the most accurately predicted class. For all models, the two most challenging emotions to classify were the happy and the fear emotion.

Overall, the models developed in this work yielded classification performances typical of speech emotion recognition models in the literature. For example, authors in [65] used the Logistic Model Tree (LMT) classifier and the lower thirteen MFCCs as features and achieved a classification accuracy of 67.14% when using the RAVDESS dataset to train and test their model. Similarly, authors in [66] managed to get a classification accuracy of 64.48% on the RAVDESS corpus using the GResNet classifier and spectrogram images. The models used in this work did not learn the emotions well enough to have a higher classification score. It could be because the models were trained only on 20,866 training samples. It could also be because of the low number of features used in both feature sets – a potential underfitting case. Using only Python for this thesis limited the number of features extracted from the audio data since there are very few Python libraries available for audio processing, and the available ones are not very rich in terms of functionality.

Creating a Multimodal Emotion Recognition System

Once all the analysis was done, the SVM, MLP, and RNN models created using the custom feature set, and the *Complete_Clean_Noise* corpus were saved to being used for real-time speech emotion recognition. A second model was saved, containing the SVM, MLP, and RNN models created using the custom feature set and the *Complete_Clean* corpus. This was done so that the first model can be used in outdoor settings, while the second model can be used indoors. If the speech emotion recognition system is implemented in a mobile application, it will allow the user to pick either of the two models based on their surrounding environment.

Ensemble learning is a branch of machine learning that refers to combining multiple machine learning algorithms to make a prediction. There are several methods of ensemble learning. The voting technique was applied to join the SVM, MLP, and RNN models. In voting, the input data is fed to all the classifiers, and predictions are made separately. The class, which was predicted the most, is selected as the final prediction of the system.

Along with the three machine learning models, the means and standard deviations of all the features were also saved. These parameters were then used for scaling the features of the input. When an utterance is recorded, the system first performs feature extraction by extracting the low-level descriptors. It then applies the required functionals to create the audio features. All audio features are standardized using the means and standard deviations of the features from training. After that, the scaled low-level descriptors are fed to the RNN model, while the scaled audio features are fed to the SVM and MLP algorithms. Once the models have made the predictions, the most common class among the three predictions is classified as the input's final emotion label. If all three models predict different classes, the MLP model's performance is selected as the final prediction since it had the highest precision score. The Python program is written so that three-second audio is recorded continuously and fed to the speech emotion recognition system when it is running. The user can stop the code anytime by pressing a keyboard interrupt. The pseudo-code for the speech emotion recognition system is given below:

```
Run a loop indefinitely until user presses keyboard interrupt:
    Record microphone audio for three seconds
    Extract low-level descriptors from audio
    Apply functionals to low-level descriptors
    Standardize low-level descriptors with mean and standard deviation from training
    Standardize output of functionals with mean and standard deviation from training
    Feed low-level descriptors to RNN
    Feed output of functionals to SVM and MLP
    Get predictions from all three models
    Compute mode of all three prediction labels
    If no mode available:
        Display emoji corresponding to the predicted label of MLP model
    If mode exists:
        Display emoji corresponding to the mode of the three predicted labels
    Repeat
```

Table 22 compares the ensemble learning model with the individual classifier models trained on the *Complete_Clean_Noise* dataset using the custom feature set. The same test set was used. It can be seen that the ensemble model achieved 0.8 % higher accuracy than the MLP model, which had the best overall scores. This 0.8 % increase translates to twenty more accurately classified samples out of the 2,607 test samples.

Table 22. Test accuracies of the three individual classifier models and the ensemble learning model. All models were trained on the Complete_Clean_Noise dataset using the custom feature set.

Model	***Test Accuracy***
SVM	66.1 %
MLP	65.7 %
RNN	63.7 %
Ensemble	66.5 %

The final step to completing this thesis work was integrating the speech emotion recognition system with a facial expression recognition system. The pre-trained facial expression recognition model was imported into the Python program that contained the speech emotion recognition model. The facial expression recognition model uses the computer's webcam to get the image data from the user. The Python script was written so that majority voting was used to predict the facial expression recognition model, and the predictions made by the SVM, MLP, and RNN models trained on the *Complete_Clean_Noise* and custom feature dataset to decide the final prediction of the model. This process is shown in the flow diagram of Figure 48.

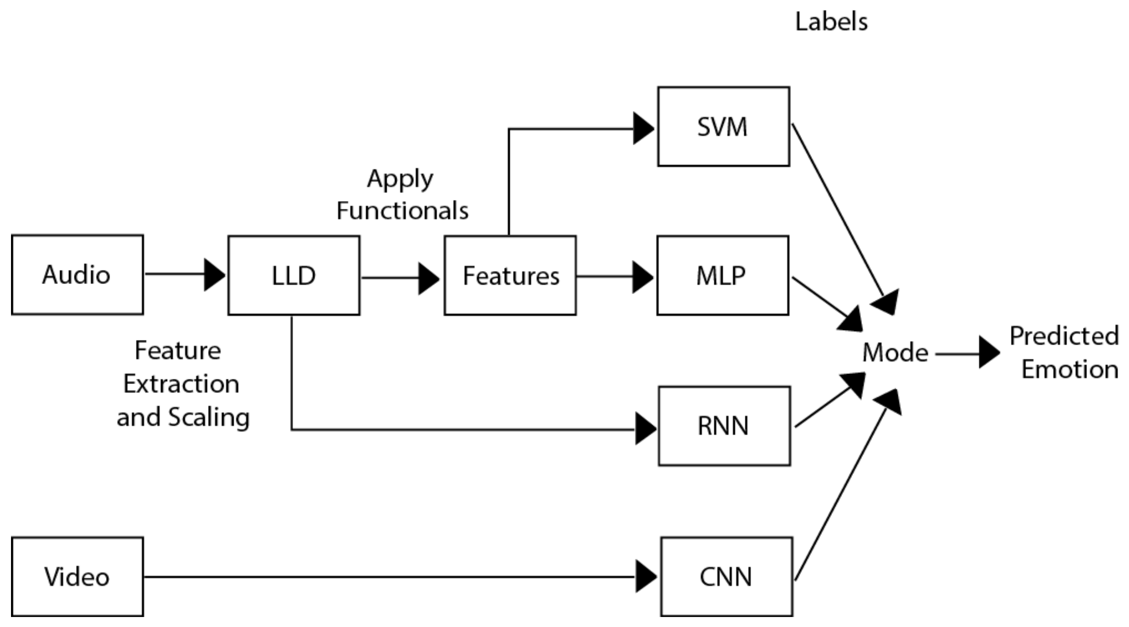


Figure 48. Flow diagram of the multi-modal emotion recognition system.

If this multimodal emotion recognition system is implemented on a mobile application, the user can choose which modality to use. The user can wish to use both models at once, which will improve predictions' confidence. The user can also choose either one of the two models. For example, the facial expression recognition model might perform better when the communication partner is not audible due to excessive background noise. The speech emotion recognition model might be a better choice when the communication partner is not visible due to poor lighting conditions. Figure 49 shows the schematic of the approach that was taken to implement the multi-modal system.

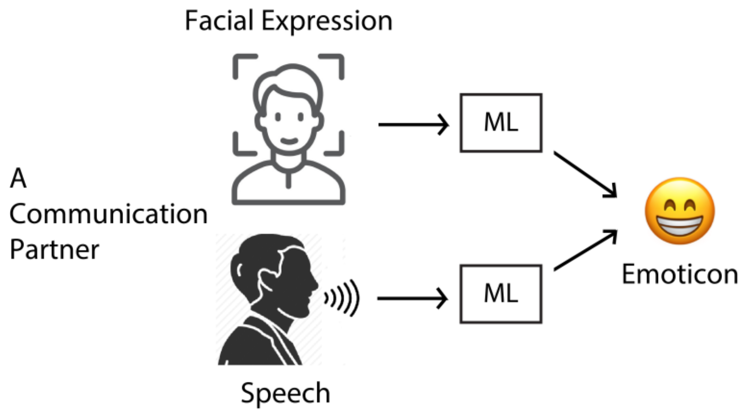


Figure 49. Schematic of the multimodal emotion recognition system.

VII. CONCLUSION

In this thesis, a speech emotion recognition solution was created for helping children with autism spectrum disorder (ASD) identify emotions in social interactions. Children with ASD have difficulty identifying emotional cues in social interactions. The objective was to develop a tool that could help these children better detect emotions when conversing with people around them. The speech emotion recognizer was developed in Python using ensemble learning, a technique used to combine multiple machine learning algorithms to get a more accurate prediction.

Three machine learning algorithms were used – a support vector machine (SVM), a multilayer perceptron (MLP), and a recurrent neural network (RNN). The datasets used to train these algorithms include the Ryerson Audio-Visual Database of Emotional Speech and Songs (RAVDESS), the Toronto Emotional Speech Set (TESS), the Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D), and the noise-added versions of the three datasets. Three noise samples were selected - a noise file containing a recording of children playing in the background, a noise file containing a recording of shopping mall ambiance, and a noise file containing a recording of cars passing by on the streets. Each clean speech utterance was added to one of the three types of noise files using MATLAB, in one of three SNR values – 0 dB, 5 dB, or 10 dB.

The final dataset contained an equal number of clean speech and noise-added files. Two separate audio feature sets were studied in this work. One feature set comprised of audio features that were handpicked based on their performances on the RAVDESS dataset, and the other feature set contained some of the features from the Geneva Minimalist Acoustic Parameter Set (GeMAPS). The performances of the models created using these two different feature sets were compared. The models developed using the custom feature set outperformed the models developed using the partial GeMAPS feature set. Therefore, the customer feature set models were used in order to construct the speech emotion recognition solution.

Two separate speech emotion recognition models were developed – one to be used indoors and the other to be used outdoors. The model created for indoor use was trained on only clean speech data from all three datasets, and the model created to be used outdoors was trained on the final dataset, which included clean speech and noise-added files from all three datasets. This was done so that if the speech emotion model was implemented on a mobile application, users could select the model they want to use based on their environment. Finally, a multimodal emotion classifier was created by joining the speech emotion recognition model with a facial expression recognition model. This produced four emotion recognition classifiers – three speech emotion recognition classifiers and one facial expression recognition classifier. The Python program was written so that if predictions from the four classifiers are unique, the facial expression recognition solution's prediction would be used, as it had better classification accuracy than the speech emotion recognition models.

Future Work

There are specific techniques that could be used to improve the speech emotion recognition system's performance. The most effective technique is to gather more data. The more data is used in training a machine learning model, the more variations in data samples are experienced and learned by the machine. However, it is essential to collect properly labeled data, as data that are wrongly labeled can worsen the model performance. The three datasets used in this work were easily accessible to the public, free of cost. However, most speech emotion corpus is not easily accessible, as they require permission from the creators or some fee. Plus, there is a limited number of North American speech emotion recognition datasets. Therefore, gathering more data is a challenging task. This work did not include any speech corpora containing recordings of children; such datasets could be used to train the speech emotion recognition system. Another way to get better prediction accuracies is to utilize more features for the design, development, and training of the deep learning model. In most of the models used in this work, the classification accuracies did not exceed 70%. Using more features will increase the machine's learning capability and improve the classification accuracy, given that the current features are incapable of learning all the complexities of the data. However, there is a risk of overfitting the model to the training data when using more features. Thus, more features should be added with caution.

Lastly, the speech emotion recognition system and the multimodal emotion recognition system can be implemented in mobile applications. The application will allow parents of children with ASD to download and use these applications on their favorite portable devices.

REFERENCES

[1] N. H. Frijda, *The Emotions*, Cambridge, England, UK: CUP, 1986.

[2] O. Korn, L. Stamm, and G. Moeckel, “Designing authentic emotions for non-human characters. A study evaluating virtual affective behavior”, *Designing Interactive Systems*, pp. 477-487, Jun 2017.

[3] A. Mehrabian, *Silent Messages*, Belmont, CA, USA: Wadsworth Pub. Co, 1971.

[4] P. Ekman and W. V. Friesen, “Constants across cultures in the face and emotion”, *Journal of Personality and Social Psychology*, vol. 17, no. 2, pp. 124–129, Feb 1971.

[5] “Autism Spectrum Disorder,” [Online] Available: https://www.nimh.nih.gov/health/topics/autism-spectrum-disorders-asd/index.shtml. [Accessed: 18-Oct-2020].

[6] S. Schelinski and K. V. Kriegstein, “The relation between vocal pitch and vocal emotion recognition abilities in people with autism spectrum disorder and typical development,” *Journal of Autism and Developmental Disorders*, vol. 49, pp. 68-82, Jul 2018.

[7] “Autism Spectrum Disorder,” [Online] Available: https://www.claytonbehavioral.com/autism-spectrum-disorder. [Accessed: 18-Oct-2020].

[8] V. T. Setty, “Speaker recognition using deep neural networks with reduced complexity”, Master’s Thesis, Engineering, Texas State University, San Marcos, TX, USA, 2018.

[9] S. Sadhu, R. Li, and H. Hermansky, “M-vectors: sub-band based energy modulation features for multi-stream automatic speech recognition,” *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6545- 6549, May 2019.

[10] W. Liu et al., “State-time-alignment phone clustering based language-independent phone recognizer front-end for phonotactic language recognition,” *14th Int. Conf. on Computer Science & Education*, pp. 863-867, Aug 2019.

[11] G. Shanmugasundaram, S. Yazhini, E. Hemapratha, and S. Nithya, “A comprehensive review on stress detection techniques,” *International Conference on System, Computation, Automation and Networking*, pp. 1-6, Mar 2019.

[12] L. Vidrascu and L. Devillers, “Detection of real-life emotions in call centers,” *9th European Conf. on Speech Communication and Technology*, pp. 1841-1844, 2005.

[13] A. K. Oryina and A. O. Adedolapo, “Emotion recognition for user centred e-learning,” *40th Annual International Computer Software and Applications Conference*, vol. 2, pp. 509-514, 2016

[14] X. Huahu, G. Jue, and Y. Jian, “Application of speech emotion recognition in intelligent household robot,” *International Conference on Artificial Intelligence and Computational Intelligence*, vol. 1, pp. 537-541, 2010.

[15] N. Kurpukdee, S. Kasuriya, V. Chunwijitra, C. Wutiwiwatchai and P. Lamsrichan, "A study of support vector machines for emotional speech recognition," *2017 8th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES)*, pp. 1-6, 2017.

[16] A. Meftah, Y. Alotaibi and S. Selouani, "Emotional speech recognition: A multilingual perspective," *2016 International Conference on Bio-engineering for Smart Technologies (BioSMART)*, pp. 1-4, 2016.

[17] G. Trigeorgis, F. Ringeval, R. Brueckner, E. Marchi, M. A. Nicolaou, B. Schuller and S. Zafeiriou, "Adieu features? End-to-end speech emotion recognition using a deep convolutional recurrent network," *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5200-5204, 2016.

[18] J. Han, Z. Zhang, F. Ringeval and B. Schuller, "Prediction-based learning for continuous emotion recognition in speech," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5005-5009, 2017.

[19] S. E. Eskimez, K. Imade, N. Yang, M. Sturge-Apple, Z. Duan and W. Heinzelman, "Emotion classification: How does an automated system compare to Naive human coders?," *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2274-2278, 2016.

[20] B. Schuller, E. Marchi, S. Baron-Cohen, A. Lassalle, H. O'Reilly, D. Pigat, P. Robinson, I. Davies, T. Baltrusaitis and M. Mahmoud, "Recent developments and results of ASC-Inclusion: An Integrated Internet-Based Environment for Social Inclusion of Children with Autism Spectrum Conditions," *IDGEI*, 2015 (No pagination provided).

[21] "Digital Audio Basics: Sample Rate and Bit Depth," [Online] Available: https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html. [Accessed: 18-Oct-2020].

[22] "Audio bit depth," [Online] Available: https://en.wikipedia.org/wiki/Audio_bit_depth. [Accessed: 18-Oct-2020].

[23] "How do i calculated the number of overlapping frames an given audio file has?" [Online] Available: https://math.stackexchange.com/questions/2249977/how-do-i-compute-thenumber-of-overlapping-frames-an-given-audio-file-has. [Accessed: 18-Oct-2020].

[24] "3. Signal Windowing," [Online] Available: http://www.atx7006.com/articles/dynamic_analysis/windowing. [Accessed: 18-Oct-2020].

[25] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences", *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 28, pp. 357–366, 1980.

[26] "Mel Frequency Cepstral Coefficient (MFCC) tutorial," [Online] Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. [Accessed: 18-Oct-2020].

[27] “Fundamental Frequency, Pitch, F0,” [Online] Available: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-73003-5_775. [Accessed: 18-Oct-2020].

[28] “Sound Intensity and Loudness,” [Online] Available: https://www.nps.gov/teachers/classrooms/sound-intensity-and-loudness.htm. [Accessed: 18-Oct-2020].

[29] “What are formants?” [Online] Available: https://person2.sol.lu.se/SidneyWood/praate/whatform.html. [Accessed: 18-Oct-2020].

[30] “Harmonicity,”, [Online] Available: https://www.fon.hum.uva.nl/praat/manual/Harmonicity.html. [Accessed: 18-Oct-2020].

[31] M. Farrús, J. Hernando and P. Ejarque, “Jitter and shimmer measurements for speaker recognition,” *8th Annual Conference of the International Speech Communication Association, Interspeech*, vol. 2, pp. 778-781, 2007.

[32] “What is HTK?” [Online] Available: http://htk.eng.cam.ac.uk/ [Accessed: 09-Nov-2020].

[33] D. Jiang, L. Lu, H. Zhang, J. Tao and L. Cai, "Music type classification by spectral contrast feature," *Proceedings. IEEE International Conference on Multimedia and Expo*, vol. 1, pp. 113-116, 2002.

[34] O. Agcaoglu, B. Santhanam and M. Hayat, “Improved spectrograms using the discrete Fractional Fourier transform,” *IEEE Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE)*, pp. 80-85, 2013.

[35] “librosa,” [Online] Available: https://librosa.org/doc/latest/index.html. [Accessed: 18-Oct-2020].

[36] A. A. Bashit, “A comprehensive solar powered remote monitoring and identification of Houston Toad call automatic recognizing device system design”, Master’s Thesis, Engineering, Texas State University, San Marcos, TX, USA, 2019.

[37] A. A. Bashit and D. Valles, "A mel-filterbank and MFCC-based neural network approach to train the Houston Toad call detection system design," *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 438-443, 2018.

[38] A. A. Bashit and D. Valles, "MFCC-based Houston Toad call detection using LSTM," *2019 IEEE International Symposium on Measurement and Control in Robotics (ISMCR)*, pp. 1-6, 2019.

[39] A. A. Bashit and D. Valles, "A solar powered raspberry pi Houston Toad call detection system using neural network model," *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 1024-1027, 2018.

[40] F. Eyben, K. R. Scherer, B. W. Schuller, J. Sundberg, E. Andre, C. Busso, L. Y. Devillers, J. Epps, P. Laukka, S. S. Narayanan and K. P. Truong, "The Geneva minimalistic acoustic arameter set (GeMAPS) for voice research and affective computing," *IEEE Transactions on Affective Computing*, vol. 7, pp. 190-202, 2016.

[41] B. Schuller, K. Scherer, M. Mortillaro, A. Batliner, F. Weninger, F. Eyben, E. Marchi, S. Steidl, A. Vinciarelli, H. Salamin, A. Polychroniou, F. Valente, S. Kim, F. Ringeval and M. Chetouani, "The INTERSPEECH 2013 computational paralinguistics challenge: Social signal, conflict, emotion, autism," *14th Annual Conference of the International Speech Communication Association*, pp. 148-152, 2013.

[42] "openSMILE audio feature extraction," [Online] Available: https://www.audeering.com/opensmile/. [Accessed: 18-Oct-2020].

[43] Y. Jadoul, B. Thompson and B. de Boer, "Introducing Parselmouth: A Python interface to Praat." *Journal of Phonetics*, vol. 71, pp. 1–15, 2018.

[44] "Praat: doing phonetics by computer," [Online] Available: https://www.fon.hum.uva.nl/praat/. [Accessed: 18-Oct-2020].

[45] M. I. Haque, "A facial expression recognition application development using deep convolutional neural network for children with autism spectrum disorder to help identify human emotions," Master's Thesis, Engineering, Texas State University, San Marcos, TX, USA, 2019.

[46] 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, pp. 1-35, 2018.

[47] "The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)," [Online] Available: https://zenodo.org/record/1188976. [Accessed: 18-Oct-2020].

[48] "Toronto emotional speech set (TESS)," [Online] Available: https://tspace.library.utoronto.ca/handle/1807/24487. [Accessed: 18-Oct-2020].

[49] "Toronto emotional speech set (TESS) A dataset for training emotion (7 cardinal emotions) classification in audio," [Online] Available: https://www.kaggle.com/ejlok1/toronto-emotional-speech-set-tess. [Accessed: 18-Oct-2020].

[50] C. Houwei, D. G. Cooper, M. K. Keutmann, R. C. Gur, A. Nenkova and R. Verma, "CREMA-D: Crowd-Sourced Emotional Multimodal Actors Dataset," *IEEE Transactions on Affective Computing*, vol. 5, pp. 377-390, Jan 2014.

[51] "CREMA-D (Crowd-sourced Emotional Mutimodal Actors Dataset)," [Online] Available: https://github.com/CheyneyComputerScience/CREMA-D. [Accessed: 18-Oct-2020].

[52] "SoundBible.com," [Online] Available: http://soundbible.com/. [Accessed: 18-Oct-2020].

[53] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, pp. 273-297, 1995.

[54] S. Raschka and V. Mirjalili, *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow*, 2nd ed., Birmingham, UK: Packt Pub., 2017.

[55] “Artificial neuron,” [Online] Available: https://en.wikipedia.org/wiki/Artificial_neuron. [Accessed: 18-Oct-2020].

[56] W. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[57] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychological review*, vol. 65, pp. 386-408, 1958.

[58] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, pp. 1735-1780, 1997.

[59] “Bias-variance tradeoff,” [Online] Available: https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff. [Accessed: 18-Oct-2020].

[60] “LEAP - High Performance Computing Cluster,” [Online] Available: https://doit.txstate.edu/rc/leap.html. [Accessed: 18-Oct-2020].

[61] “Technology,” [Online] Available: http://hipe.wp.txstate.edu/technology/. [Accessed: 18-Oct-2020].

[62] K. H. Lee, H. K. Choi, B. T. Jang and D. H. Kim, “A study on speech emotion recognition using a deep neural network,” *International Conference on Information and Communication Technology Convergence (ICTC),* pp. 1162-1165, 2019.

[63] F. A. Shaqra, R. Duwairi and M. Al-Ayyoub, “Recognizing emotion from speech based on age and gender using hierarchical models,” *Procedia Computer Science*, vol. 171, pp. 37-44, 2020.

[64] “The number of hidden layers,” [Online] Available: https://www.heatonresearch.com/2017/06/01/hidden-layers.html. [Accessed: 22-Oct-2020].

[65] A. A. A. Zamil, S. Hasan, S. M. Jannatul Baki, J. M. Adam and I. Zaman, “Emotion detection from speech signals using voting mechanism on classified frames,” *International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, pp. 281-285, 2019.

[66] Y. Zeng, H. Mao, D. Peng and Z. Yi, “Spectrogram based multi-task audio classification*,” Multimedia Tools and Applications: An International Journal*, vol. 78, no. 3, pp. 3705-3722, 2019.