Speech Emotion Detection using Deep Learning

Name: Hamed Vaheb
Course: Prototyping with Deep-Learning

Abstract

One of the most essential means of expressing emotions among humans is through speech. Emotion recognition from speech signals is a prominent yet challenging task of Human-Computer Interaction (HCI). Deep learning techniques have been recently proposed as an alternative to traditional models in this field. This project uses a speech emotion recognition system, for which three types of the Recurrent Neural Network (RNN) models were used, i.e., gated recurrent unit (GRU), bidirectional GRU (BIGRU), and long short-term memory (LSTM). The models classify emotional state of a given speech audio files. The dataset that were used to train the models was RAVDESS. Mel-frequency cepstral coefficients (MFCCs) of the speech signals are extracted, concatenated, stored and then are fed into the models. Various strategies were employed to improve our initial results, e.g., early stopping, feature selection, etc. The results show that the models are capable of classifying the emotions much stronger than a random system.

1 Introduction

"Previously, a series of studies rooted in psychology rather than in computer science investigated the role of acoustics of human emotion. Even the most primitive can recognize the tones of love and fear and anger; and this knowledge is shared by the animals. The language of the tones is the oldest and most universal of all our means of communication. It appears the time has come for computing machinery to understand it as well." [17]

The emotional state of human beings is an important factor in their interactions, influencing most channels of communication such as facial expressions, voice characteristics, and the linguistic content of verbal communications. Moreover, it affects how an agent (either human or computer) judges an interaction. For instance, simply writing "go home", without producing any accompanying voice intonation or facial expression, could be understood as an imperative statement of very positive or very negative polarity: e.g., if a child is feeling sick and the teacher sends him home; or a child that is not behaving well in the classroom and the teacher sends him home. With facial expressions and intonations humans are communicating emotions, which could be related to judgments and subjective perception. Emotions influence judgment by how one feels about the object that affects judgment [12].

Speech is one of the primary faucets for expressing emotions, and thus for a natural human-machine interface, it is important to recognize, interpret, and respond to the emotions expressed in speech. Emotions influence both the voice characteristics as well as linguistic content of speech [14].

Speech emotion recognition (SER) is an important but challenging component of Human-Computer Interaction (HCI), and more specifically, that of Communicating with Computers (CwC). Alexa, Cortana, Siri, and many more dialogue systems are examples of communication with computing machinery. There are numerous studies aiming to enrich AI systems with emotional intelligence abilities by extracting emotions from words (sentimental analysis) and voices (SER) [17]. One of the successful publicly used examples of SER is European ASC-Inclusion project that is dedicated to developing video game that teaches autistic children how to express their emotions. Intriguingly, voice-only as modality worked more efficient than video-only or audiovisual communication according to a recent study [9].

In contrast with traditional devices and methods, which aimed at detecting emotions with regard to verbal content of the voices, modern systems concentrate directly on voice interactions and extract features from voice, even those that may go unnoticed or exceed human's hearing abilities.

In the literature of speech emotion recognition (SER), as numerous models have been employed, we can consider them as traditional and modern. Among traditional ones, there lie many well-established speech analysis and classification techniques. Deep Learning techniques are regarded as among modern models, which have several advantages over traditional methods, including "*their capability to detect the complex structure and features without the need for manual feature extraction and tuning; tendency toward extraction of low-level features from the given raw data, and ability to deal with un-labeled data*". [6]. This distinction is depicted in the figure 1.
The questions that are aimed at tackling in this project are as following:

1. Are neural networks capable of detecting emotions from audio files?

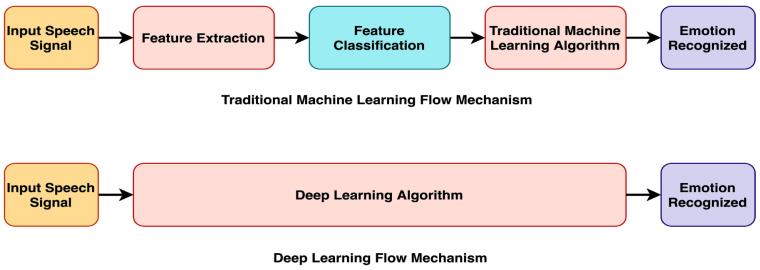


Figure 1: Comapring DL models and traditional ones. Figure is from [6]

2. What features of the sound are the most important ones in determining the emotions?
3. Among main recurrent neural network architectures, i.e., LSTM, GRU, BIGRU, all of which were used in the project, which one have the best performance in classifying emotions from audio files?
4. What are advancements and limitations of this project?

The models LSTM, GRU, and BIGRU models were employed, all of them consisting of an input layer, two layers of the specific archeitecture of model (LSTM, GRU, or BIGRU accordingly), and their last layer is the "softmax" activation function.
The models were trained on the RAVDESS dataset, from which only speech files were used and song files were discarded. MFCCs of the audio signals (using the `Librosa` package), concatenated, and stored for the training and evaluation (testing) of our model.
The detailed stages of the implementation plan and the detailed explanation of each stage is elaborated in the section 3.
In what follows, related works are mentioned and it is specified how they are related to this project.

2 Related Works

Numerous works have used deep learning for SER. The most notorious ones are explained in the following:
[14] employed a biderctional LSTM that earns both the short-time frame-level acoustic features that are emotionally relevant, as well as an appropriate temporal aggregation of those features, while introducing a strategy for feature pooling over time which uses local attention in order to focus on specific regions of a speech signal that are more emotionally salient. Combining the LSTM with pooling focus on emotionally salient parts of a sentence. With the attention model, the network can simultaneously ignore silence frames and other parts of the utterance which do not carry emotional content. However, The weighted pooling with attention model increase accuracy slightly.

[10] used a bidirectional long short-term memory (BLSTM) while employing a maximum-likelihood based learning process in order to extract high-level representation of emotional states with regard to its temporal dynamics. In conventional algorithms, all frames in the utterance are mapped to the same label, whereas this work considers the emotional state as a random variable, which is more realistic and led to improved performance. However, the distribution of the emotional state might not be readily possible to be obtained in various datasets and settings. Therefore, generalizing the used approach might be difficult. Moreover, the local correlational features have been neglected. In our work, since we use RNNs, long-term features are also considered.

[15] used a two-stream deep convolutional neural network with an iterative neighborhood component analysis (INCA) to learn mutually spatial-spectral features and then selects the most discriminative optimal features for the final prediction. Using two channels for the network, the model extracts features from spectral domain using the first channel, and then uses the second one to extract features from the spatial domain, which makes it more equipped compared to models that use only one space. Although the purpose of INCA is stated as removing redundancy and discrepancy from the fused features, it is not clear that what would be the actual difference if they did not use this algorithm. For the practical implementation of the system in a real-time environment, a high computational devices and more natural datasets with a huge amount of data is needed. An improvement possibility is extending and adapting the developed two-stream model to an automatic speaker recognition system and evaluated it over a more natural and huge corpora for real-time performance in industrial Our model is also is limited in the sense that it is not able to detect emotions on the fly.

[5] used `Librosa` audio library to extract features (mel-frequency cepstral coefficients, chroma-gram,

mel-scale spectrogram) from sound files and used them as inputs for the one-dimensional CNN. They also employed cross-validation which enhance generalization ability of the model. Using a third party package reduces the work required for extracting features and also makes the work reproducible. The model confidently identifies the strong emotions like "angry". However, it confuses some close emotions like "calm" and "sad" or "happy" and "surprised". The order of stacking sound features play an important role in the final performance, yet the order used was obtained experimentally and without a methodical approach or explanation, and therefore the order might not be optimal. We will scrutinize the order of stacking, and if possible, try all different combinations in order to be more methodical about it. We also use `Librosa` audio library to extract features, which is explained in more detail in section 3.5.

[18] uses two convolutional neural networks and long short-term memory (CNN LSTM) networks, one 1D CNN LSTM network and one 2D CNN LSTM network, were constructed to learn local and global emotion-related features from speech and log-mel spectrogram respectively. A hybrid model is used which extracts extract long-term contextual dependencies using its LSTM component, and also learns local correlations using convolutional and max-pooling layers. However, no data augmentation is used for reducing the noise and also to enhance the generalization.

We used both speech files to analyze and classify the emotions. Some works have used speech songs files and also videos in addition as well.
The stages of our implementation plan the detailed explanation of each stage is elaborated in the following sections:

3 Methodology

The parts of this sections are organized as follows: In 3.1 Modeling Emotions, the model of representing the emotions is explained, which is through discrete classes. In 3.2 Packages, the packages used in this project are specified, as well as the methods and objects we used from these packages. In 3.3 Data, At first describe the RAVDESS dataset is described, then elaborated on how audio files are loaded from this dataset. Finally, waveplot and spectogram of samples of the data are visualize for illustration purposes. In 3.4 Preprocessing, at first it is explained how labels are encoded and decoded, then it is focused on the audio effects applied on the data, which are normalizing, trimming silence, padding, and reducing noise. In 3.5 Feature Extraction, the feature that is extracted from audio signals, i.e., Mel-frequency cepstral coefficients (MFCCs) are explained. In 3.7 Models' Architecture: LSTM, GRU, BIGRU, we provide details about the RNN models implemented, and also the strategies and techniques employed for controlling the train and test error and improving them.

3.1 Modeling Emotions

Approaching the automatic recognition of emotion requires an appropriate emotion representation model. This begs the question: How to represent emotions per s? Two models are usually found in practice. The first model is discrete classes, such as the Ekman "big six" emotion categories, including anger, disgust, fear, happiness, and sadness—often added by a "neutral" rest-class as opposed to a value "continuous" dimension approach. Although the latter approach has garnered more attention, the former is used in thie work, for simplicity. And the labels of the dataset has an additional "surprise" emotion compared to the big six. Consequently, our labels will be

{neutral, calm, happy, sad, angry, fear, disgust, surprised}

In 3.4, we will explain how we encode and decode our labels to numbers and vice versa.

3.2 Packages

In what follows, the packages we used and more specifically, the methods and objects we used from them are explained:

`librosa`: Used for capturing sampling rate of each audio file 3.3.2, for processing and more specifically, applying effects on the files 3.4.2, and for extracting features from the processed files 3.5. `librosa` is a python package for music and audio analysis, which provides the building blocks necessary to create music information retrieval systems. For advanced introduction to this package and its design principles, motivations, etc., please refer to [13].

`pydub`: Used for opening each WAV file 3.3.2, and applying effects on the files 3.4.2. Pydub is package for manipulating audio with a simple and easy high level interface.

3.3 Data

3.3.1 Description

The datasaet used for this work is **RAVDESS** [11] database. The database is gender balanced consisting of 24 professional actors, vocalizing lexically-matched statements in a neutral North American accent. There are both speech and songs in the data, however, for our purpose the songs were discarded, as they are not pertinent to this work. Audio files contain the following emotions: calm, happy, sad, angry, fearful, surprise, and disgust. Each emotion is produced at two levels of emotional intensity, with an additional neutral expression. All conditions are available in three modality formats: Audio-only, Audio-Video, and Video-only (no sound). This work uses only speech data (but only audio versions) and processed them as explained in the next section. The audio-only format has the following properties: (16bit, 48kHz .wav) The frequency of each label is plotted in figure 2.

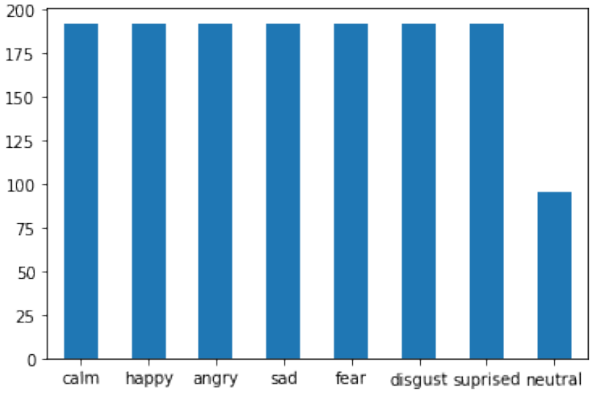


Figure 2: Histogram of Emotions (Labels)

As the neutral label has half number of samples compared to other labels, this label was discarded so that all the labels would be balanced.

3.3.2 Data Loading and Visualization

Ravdess dataset comprise WAV files. We feed path of each audio file to the `librosa.load` function, which loads audio file as a floating point time series. This function has an argument `sr` which we set as `None` so as to preserve the native sampling rate of the file.

3.4 Preprocessing

The preprocessing stages are explained in the following. In figure 3, all the stages are included.

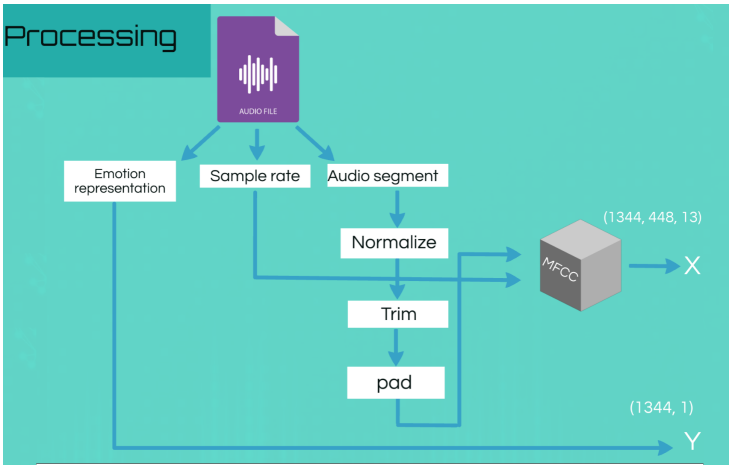


Figure 3: Illustration of outline of preprocessing and feature extraction

3.4.1 Encoding and Decoding Labels

In Ravdess dataset, each sample has a format akin to `"03-01-04-01-01-01-09.wav"`, from which the third number (lying at 6-8 position of the file if we start at zero), indicates the emotion. Thus, the functions `emotion_encode` are implemented, which extracts the number associated with the emotion from each path and encodes it to a number in $\{0, 1, ..., 7\}$, with the following structure:
0: neutral, 1: calm, 2: happy, 3: sad, 4: angry, 5: fear, 6: disgust, 7: surprised Moreover, the function `emotion_label` is implemented, which takes an encoded number and outputs the type of emotions. The functions `emotion_encode` and `emotion_label` can be regarded as inverses of each other.
After encoding the labels, as they serve as our labels, they are stored in a variable Y, which is separate from our features, which is stored in X. This stage is under the term "Emotion Representation" in figure 3.

3.4.2 Audio Effects

The following effects are applied on the audio files: At first the files are normalized to `headroom = 5.0` using `pydub.effects.normalize` and then transformed to an array. `headroom` is how close to the maximum volume to boost the signal up to (specified in dB). If for instance, the maximum is 0, this audio segment is silent, and can't be normalized. Then, silence is trimmed from the beginning and the end of the files using `librosa.effects.trim` by choosing the threshold `top_db = 30`, which makes the method to consider below this threshold reference as silence. Thereafter, files are padded for constructing equally sized arrays using `keras.pad_sequences`, which is a necessary step for feeding to the RNN models. For this purpose, the maximum length of all files are found, so as to choose it as `maxlen` in the `pad_sequences` method.
Finally, noise from the data is reduced. Although there is no noise to reduce from RAVDESS databases, `reduce_noise` function by `noisereduce` library attributes a uniform stamper to the audio files, hence it is used in thie work.
The aforementioned audio effects are under the terms "Normalize", Trim", and "Pad" in the figure 3. For illustrating the shape of audio files after each effect, figure 4 is provided.

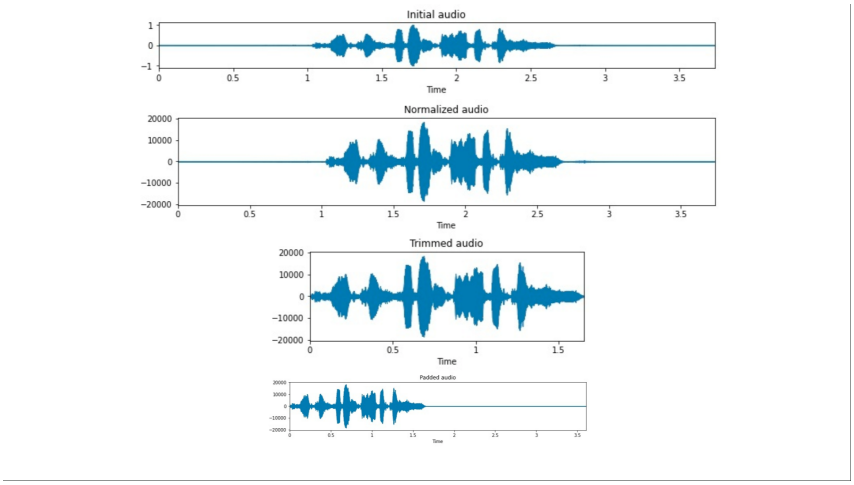


Figure 4: Audio Effects

3.5 Feature Extraction

3.5.1 Categorizing Audio Features

There are multiple categorizations for audio features, e.g., levels of abstraction, domain (either signal or time), temporal scope, etc. For more details regarding this, please refer to [8]. The sole feature used in this work was Mel-frequency cepstral coefficients (MFCCs). In what follows, the most important categorizations are briefly described, along with position of MFCCs in the each categorization.

Levels of Abstraction

The three levels of abstraction is depicted in the figure 5. Between low-level and high-level features, there is a discrepancy—commonly known as "semantic gap". In order to mitigate the problem of the semantic gap, a third class of features has been established: the so-called mid- level features, which are those that a

combination of or extension to low-level features that incorporate additional, high-level knowledge. Mid-level representations are considered closer to human concepts and should therefore allow more meaningful comparisons between entities, also carry more relevant information for machine learning approaches to generalize from. [8]
MFCCs also lies in the mid-level features, as can be seen from figure 5.

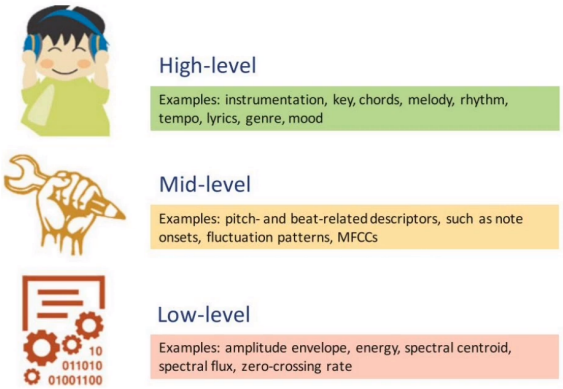


Figure 5: Levels of feature abstraction. Figure is from [8].

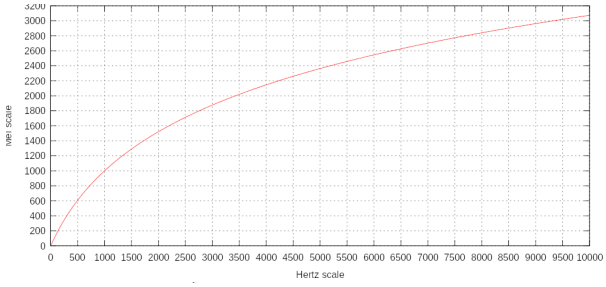
Time Domain and Frequency Domain

Time domain features are extracted from waveforms (such as the upper plot of 3.5.2, which have time and amplitude in X-axis and Y-axis respectively. Examples of these features include Amplitude envelope, Root-mean square energy, Zero crossing rate, etc. However, as many aspects of sound are characterized by frequency, we need frequency-domain features which concentrate on frequency components of sound, and are obtained by applying Fourier transforms to signals from the time-domain representation so as to transit to the frequency-domain. For instance, frequency plot can be obtained by applying Fourier transform on waveplot, as seen in the middle plot of figure 3.5.2. Examples of frequency-domain features include Band energy ratio, Spectral centroid, Spectral flux, etc.
The demerit of the two mentioned representations is that in each of them there is information either about time or frequency. However, there is also a third type of representation in which both aspects of sound are retrieved concurrently, and this representation is called Time-frequency representation. Examples of this type include Spectrogram, Mel-spectrogram, Constant-Q transform, etc.
In traditional machine learning models for speech processing, audio features were used by manually choosing relevant features from time of frequency domain. In deep learning approach, however, the feature selection is done automatically after feeding a collection of unstructured audio representations to model, which can be done by either passing raw data, or by passing the mel-spectrograms, or MFCCs, which contain information from both time and frequency domain.
As mel-spectrogram are closely related to MFCCs and they were also used as features in one of our experiment trials, they will be elaborated in the following.

3.5.2 Mel Spectrogram

Mel-spectrograms are obtained by applying short-time Fourier transform (STFT) on the waveplot. Two rescalings is also applied in the process, one is to convert frequency to mel scale, and another is to convert amplitude to decibels. Both of the rescalings will be explained in details in the following:

Mel Scale: The mel scale relates perceived frequency, or pitch, of a pure tone to its actual measured frequency. Humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features match more closely what humans hear, and hence it is perceptually relevant. The formula and plot of converting a given frequency in Hertz to mel scale is depicted in figure 6. Note that the whole procedure of converting to mel-scale requires applying mel-filterbanks to spectrogram, which is explained in details in [2].



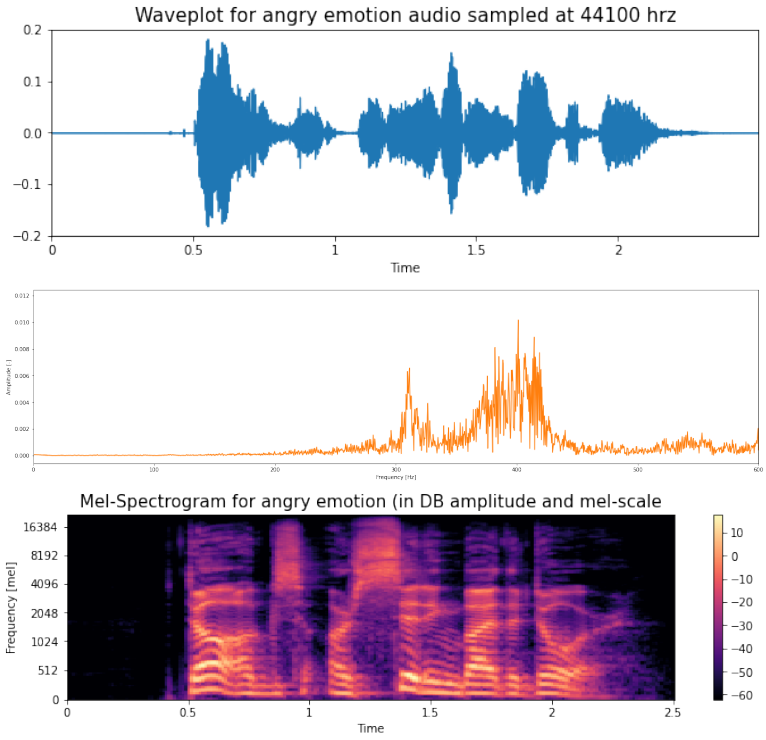
$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

Figure 6: Mel scale plot and formula

Power to Decibels: Another rescaling that is employed is to convert amplitude to Decibels. Two amplitudes are often best compared using their ratio rather than their difference. For example, saying that one signal's amplitude is greater than another by a factor of two is more informative than saying it is greater by 30 millivolts. This is true for any measure of amplitude (RMS or peak, for instance). To facilitate this, amplitude is rescaled to logarithmic units called decibels. The `librosa.power_to_db` method is used for this purpose.
The procedure of constructing a mel spectrogram is summarized in the following steps:

1. Extract Short Time Fourier Transform (STFT)
2. Convert amplitude to Decibels (DBs)
3. Convert frequencies to Mel scale

In figure 3.5.2, a waveplot and its corresponding spectrogram of a sample of the dataset is visualized. The sample is taken from the folder "Actor_01" and file name "03-01-05-01-02-02-01", which has the following characteristics: Gender: Male, Emotion: Angry, Intensity: Normal



3.5.3 MFCC (Mel-frequency cepstral coefficients)

The "mel-frequency" term in the MFCC is about the mel scale which is described earlier. Another building block of the MFCCs is cepstrum, which accounts for "cepstral coefficients" term. Cepstrum is explained in what follows.

Cepstrum The cepstrum is a representation used to convert signals combined by convolution (such as a source and filter) into sums of their cepstra, for linear separation. In particular, the power cepstrum is often used as a feature vector for representing the human voice and musical signals. Given a signal $x(t)$, a simplified formula of cesptrum $C(x(t))$ is defined in the following:

$$C(x(t)) = \overbrace{F^{-1}[log(\underbrace{F[x(t)]}_{\text{Spectrum}})]}^{\text{Cepstrum}}_{\text{Log spectrum}}$$

where F and F^{-1} are Fourier transform and inverse Fourier transform respectively. Therefore, cepstrum can be thought of as a spectrum of a spectrum. Frequency spectrum of a signal is the range of frequencies contained by a signal. In figure 7, the effect of each function involved in cepstrum is visualized. In the upper part of the figure, which we apply F on "Signal" diagram", there will be a shift from time domain to frequency domain, and the result is termed "Power spectrum". After taking the logarithm of amplitude, which will transit us from the upper right to the lower left part of the figure, we reach the "Log power spectrum" diagram, where applying F^{-1} on this diagram will lead to the last shift, from frequency domain to time domain. However, the time unit obtained at the end is not the same as the initial time domain. Rather, it is quefrency (ms). For an extensive explanation and history of quefrency and cepstrum, please refer to [16].
The significance of quefrency in speech processing is that if we formalize speech as a convolution of vocal tract frequency response with glottal pulse, then quefrency values are effective in isolating the two mentioned speech components.To have a more comprehensive detail about this and in general, theory of cepstral analysis of speech, and also about how to put them in practice, please refer to the AMRITA simulation lab [3].

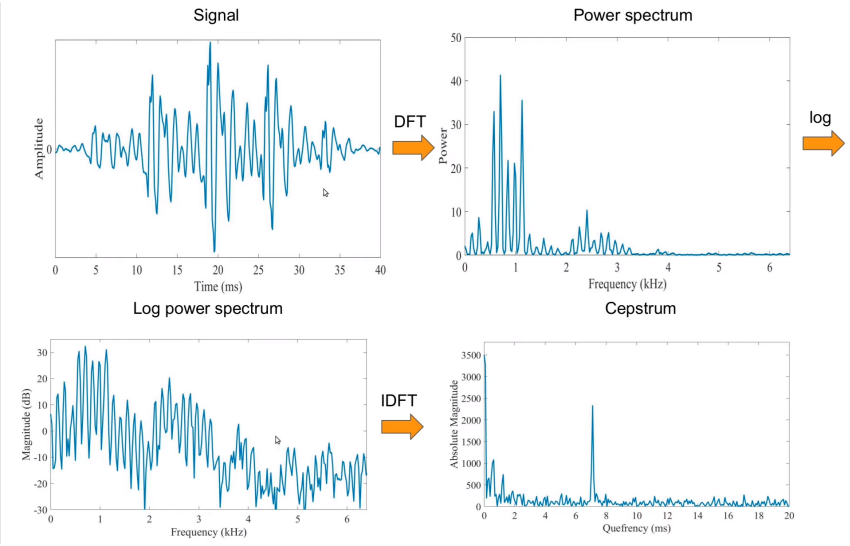


Figure 7: Cepstrum Function Components

Now that mel scale and cepstrum is explained, the computation of MFCCs can now be understood. When computing cepstrum of a signal, if before applying F^{-1}, the log-amplitude is rescaled to mel scale, and also instead of applying F^{-1}, Discrete Cosine Transform (DCT) is applied (an advantage of it is that DCT yields real numbers instead of complex ones). The summarized procedure to develop MFCCs is the following:

1. Convert from Hertz to Mel Scale.
2. Take logarithm of mel representation of audio.
3. Take logarithmic magnitude and use discrete cosine transformation.
4. This result creates a spectrum over mel frequencies as opposed to time, thus creating MFCCs.

For a through explanation of computing MFCCs, please refer to [7], and fore more information about theory of MFCCs, please refer to [1].

In figure 8, the plot of MFCCs are illustrated for the same audio sample for which mel-spectorgram was shown in 3.5.2,

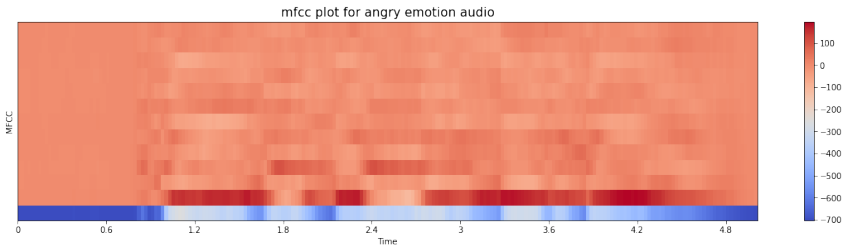


Figure 8: MFCCs for audio sample labeled with "angry"

The number of coefficients used for MFCCs (`n_mfcc`) is a hyperparameter of our model. Because of this, the number of MFCCs will vary based on the problem. Four our case, we used 13.

After processing all files, and extracting all the features from the files, they are concatenated, and then both features and labels are stored as two `json` files (X and y respectively). By dint of this, in the final step of applying the model, dataset can be loaded in order to feed them into the RNN models. This is also illustrated in figure 3.

3.6 Defining Recurrent Neural Networks

3.6.1 General RNNs

As the dataset consists of singals, which is a type of time series data, and hence sequential data, RNNs are prudent candidates among classification models. Moreover, RNNs can effectively remember relevant long-term context from the input features. Remembering long-term context is not possible in multi-layer perceptrons (MLPs) as in their structure it is often presumed that all the inputs and outputs are independent of each other. the other hand, in RNN the behavior of hidden neurons might not just be determined by the activations in previous hidden layers, but also by the activations at earlier times. Furthermore, the activations of hidden and output neurons will not be determined just by the current input to the network, but also by earlier inputs.
In a more mathematical manner, consider the classical form of a dynamical system:

$$s^{(t)} = f(s^{(t-1)}; \theta), \tag{1}$$

where $s^{(t)}$ is called the state of the system. If we unfold equation (1) for $\tau = 3$ time-steps, we obtain

$$s^{(3)} = f\big(s^{(2)}; \theta\big) \tag{2a}$$

$$= f\big(f(s^{(1)}; \theta); \theta\big). \tag{2b}$$

The unfolded equation yielded an expression devoid of recurrence that can be depicted by a traditional directed acyclic computational graph. The unfolded computational graph of equation (1) and equation (2b) is illustrated in figure 9.

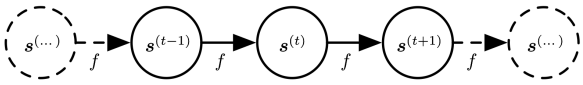


Figure 9: The classical dynamical system described by equation (1), illustrated as an unfolded computational graph. Each node represents the state at some time t, and the function f maps the state at t to the state at $t + 1$. The same parameters (the same value of θ used to parameterize f) are used for all time-steps. Figure is from [4]

To incorporate the input of each step, consider a dynamical system driven by an external signal x^t,

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta), \tag{3}$$

where we see that the state now contains information about the whole past sequence.

"Recurrent neural networks can be built in many different ways. Much as almost any function can be considered a feedforward neural network [according to universal approximation theorem], essentially any function involving recurrence can be considered a recurrent neural network." [4]

Many recurrent neural networks use equation (4) or a similar equation to define the values of their hidden units. To stress the fact that the state is the hidden neurons of the network, we now rewrite equation (3) using the variable h to represent the state,

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta), \tag{4}$$

illustrated in figure 10.

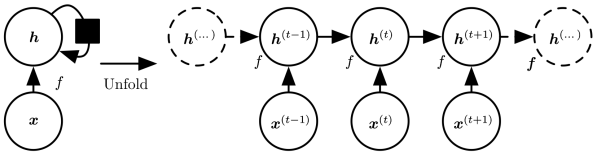


Figure 10: A recurrent network with no outputs. This recurrent network just processes information from the input x by incorporating it into the state h that is passed forward through time. (*Left*) Circuit diagram. The black square indicates a delay of a single time-step. *Right* The same network seen as an unfolded computational graph, where each node is now associated with one particular time instance. Figure is from [4]

3.6.2 Gated Recurrent Neural Network (GRNN), and Long-Short Term Memory (LSTM)

The most effective sequence models used in practical applications are called gated RNNs. These include the long short-term memory and networks based on the gated recurrent unit. Generally, as RNNs are equipped with learning long-term dependencies, they have the issue of vanishing or exploding gradients. GRNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode, which was made possible by including connection weights that may alter at each time-step. This allows them to tackle the challenge of vanishing or exploding gradients. Once that information has been used, however, it might be useful for the neural network to forget the old state. The intuition is that although remembering past state of the network (either earlier inputs or outputs) is useful, but not all states are useful, hence it would be an intelligent and more human-wise behavior that the model decides when to clear the past state, which is what GRNNs do.
The difference between a GRNN and LSTM is that GRNNs have input gate and output gate, while LSTM has an additional forget gate.
In addition to GRNN and LSTM, we also used a bidirectional network of GRU. Generally, a bidirectional RNN connect two hidden layers of opposite directions to the same output. With this form of generative deep learning, the output layer can get information from past (backwards) and future (forward) states simultaneously.

3.7 Models' Architecture: LSTM, GRU, BIGRU

After loading the stored processed data (explained at the end of 3.5), it is split into train, validation, and test sets with size proportion of sets as 70%, 20%, 10% respectively.
The result of training on the three recurrent nueral network archietectures, i.e., GRU, BIGRU, and LSTM are elaborated in the following. The data are ready now to be fed into the models defined using `Keras` package. Three RNN models were implemented, i.e., Long-short Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bidirectional GRU (BIGRU). For LSTM, `Keras.layers.LSTM` was used, and for GRU and BIGRU, `Keras.layers.GRU`, and `Keras.layers.Bidirectional` were used respectively.
The models were trained on training set, while also using the validation set. The callback functions employed were `EarlyStopping` and `ReduceLROnPlateau`, though the latter did not have any effect, as the plots of loss (provided in 4.3) did not exhibit a constant behavior at any range of points in epochs. The early stopping were used with parameters `patience=20, min_delta=0.0001`. The early stopping decides to stop after the loss evaluated on the validation set would not improve more than the determined threshold (min_delta).
Among the different setting and choice of number of neurons and layers we tested for the 3 architectures of LSTM, GRU, and BIGRU, the following were the one with the most accuracy and least error on the test set:

1. LSTM: Layers: LSTM(128), LSTM(64), Activation Function(Softmax)
2. GRU: GRU(128), GRU(64), Activation Function(Softmax)
3. BIGRU: BIGRU(128), BIGRU(64), Activation Function(Softmax)

As an example, the structure of the implemented GRU model, which outperformed the other models, is illustrated in 11.

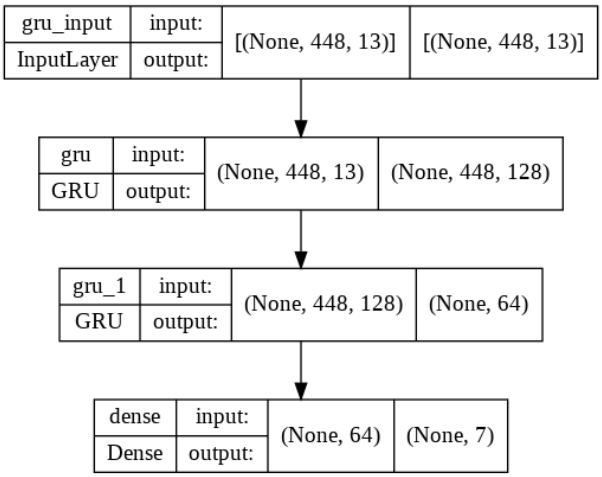


Figure 11: Implemented GRU Model's Structure

In what follows, detailes are provided on how the models train on training data and test on test data: The loss function used were categorical cross-entropy. The optimization algorithm chosen were RMSProp, which is expalined in details in 3.7.1. The choice of hyperparameters, and the techniques used for improving the generalization capability of our model are explained in 3.7.2.

3.7.1 Tuning Parameters: RMSProp

The optimization algorithm we choosed were RMSProp. In what follows, we explain this algorithm: In the standard gradient descent algorithm, it takes larger steps in one direction and smaller steps in another direction which slows down the algorithm. For instance, it takes larger steps in the y axis in comparison to x axis. Momentum resolves this issue by restricting oscillation in one direction so that algorithm converges faster. It provides the freedom to use higher learning rates with less chance of confronting the overshooting challenge. It is noteworthy to pinpoint that it employs the exponential moving average in its architecture.
In the following figures, paths taken by SGD with momentum and SGD without momentum are compared:

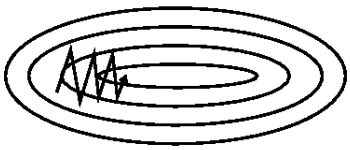


Figure 12: Path taken by SGD without Momentum

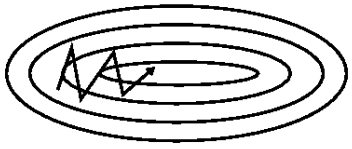


Figure 13: Path taken by SGD with momentum

3.7.2 Tuning Hyperparameters and Regularization Techniques

The hyperparameters chosen were as following: `batch_size = 23`, `epochs = 340`
We used early stopping with `patience = 10` and threshold `min_delta = 0.0001` in order to modify the number of epochs such that the model stops when the gap between train error and validation error exceeds the threshold. We save the weights of our model after the training is finished. We also used reduce learning rate with `patience = 40` although judging based on the plots of training and validation errors, this was not required. As mentioned in 3.5, `n_mfcc` is also a hyperparameter, which chosen to be 13.

4 Experimental Evaluations and Discussion

Two sets of observations were made during both training and testing phases. One set of observations were for the purpose of, or as a result of finding the optimal architecture. Another set of observations were after evaluating the optimal found architecture. Both kinds of observations are stated in 4.1. Then, using various metrics, loss and accuracy for all models are provided in 4.2, and also plot of loss and accuracy on validation and test sets, as well as plots of confusion matrix, for all models are visualized in 4.3.

4.1 Experiments and Observations

For finding the best architecture, several configurations of GRU, BIGRU, and LSTM models were tested. Throughout the trials, the following observations were made:
Standardrizing data after storing them in X and y variables, using either `MinMaxScaler` or `StandardScaler` would only worsen the accuracy. Data augmentation methods were not used, as it requires more domain knowledge, i.e., one needs to know what kind of method will not alter the relationship between the audio and the emotion (label) in the audio. Adding dropout layers did not mitigate overfitting for the performed trials.

After finalizing the model selection, choosing the best configuration, training and evaluating, the following observations are made based on the reports and plots in 4.2 4.3 respectively: GRU outperformed all the RNN models. Adding different types of features, such as those from time domain, e.g., zero crossing rate (ZCR), or even more raw representations such as melspectrograms, did not lead to significant improve.

4.2 Report Loss Metrics

The performance of RNN models were measured on both validation and test sets, using the following measures: categorical accuracy, cross entropy loss, precision, recall, fscore, ROC_AUC score. In what follows, all the measures are reported.

Accuracy and Cross Entropy Loss

Model	Validation		Test	
	Accuracy(%)	Loss	Accuracy(%)	Loss
GRU	60	1.36	61	1.39
BIGRU	57	1.27	60	1.48
LSTM	43	1.47	41	1.61

Table 1: Loss and accuracy of all models

Precision, Recall, Fscore

Precision, Recall, and Fscore are computed for both "macro" and "weighted" averages, and they are evaluated on both validation set and test set.

Validation Set

In table 4.2, all the aforementioned metrics for the three architectures evaluated on the validation:

Model	Macro			Weighted		
	Precision	Recall	Fscore	Precision	Recall	Fscore
GRU	0.61	0.6	0.6	0.61	0.6	0.6
BIGRU	0.57	0.57	0.56	0.57	0.57	0.56
LSTM	0.44	0.43	0.42	0.44	0.43	0.43

Table 2: Precision, Recall, and Fscore of all models evaluated on validation set

Test Set

In table 4.2, all the aforementioned metrics for the three architectures evaluated on the test set:

Model	Macro			Weighted		
	Precision	Recall	Fscore	Precision	Recall	Fscore
GRU	0.63	0.62	0.61	0.64	0.61	0.61
BIGRU	0.63	0.62	0.6	0.65	0.6	0.56
LSTM	0.44	0.43	0.42	0.44	0.43	0.43

Table 3: Precision, Recall, and Fscore of all models evaluated on test set

ROC AUC

In following the metric Area Under the Receiver Operating Characteristic Curve (ROC AUC) is reported. As there are multiple classes, both One-vs-One (OvO) ROC AUC scores and One-vs-Rest (OvR) ROC AUC scores were computed. Again, the scores are computed for both "macro" and "weighted" averages, and they are evaluated on test set. In table 4.2, all the ROC AUC scores are reported.

Model	ROC-AUC Score			
	Macro		Weighted	
	OvO	OvR	OvO	OvR
GRU	0.9	0.9	0.9	0.9
BIGRU	0.9	0.9	0.9	0.9
LSTM	0.79	0.78	0.78	0.77

Table 4: OvO and OvR ROC_AUC scores of all models

4.3 Plotting Loss, Accuracy, and Confusion Matrix

Loss and Accuracy

In following, the categorical crossentropy loss of all the models evaludated on the validation set per epoch are visualized.

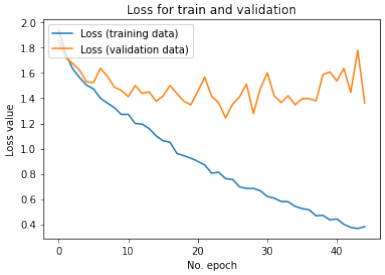


Figure 14: Loss of GRU

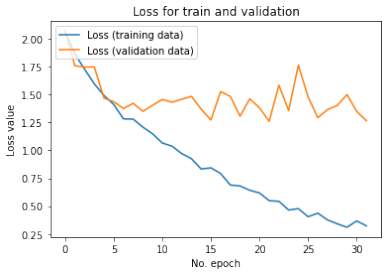


Figure 15: Loss of BIGRU

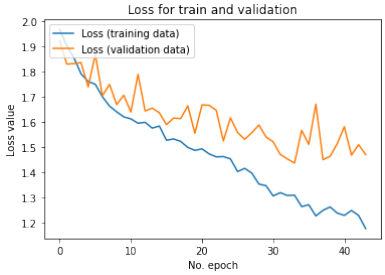


Figure 16: Loss of LSTM

In contrast, the accuracy would behave as inverse of the prior loss plots. In following, categorical accuracy of models evaluated on the validation set per epoch are visualized.

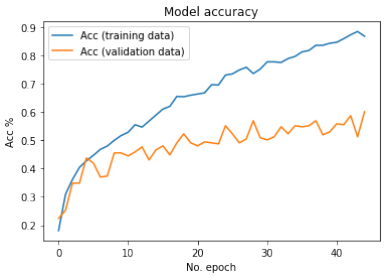


Figure 17: Accuracy of GRU

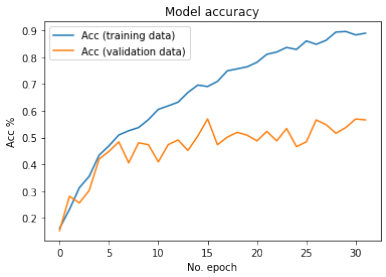


Figure 18: Accuracy of BIGRU

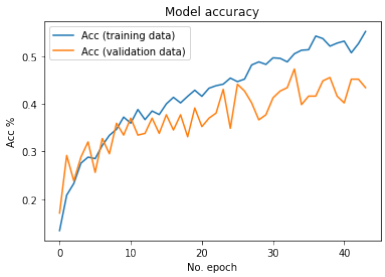


Figure 19: Accuracy of LSTM

Based on the loss and accuracy plots, all models are subject to overfitting and early stopping were used to control the behavior per epochs.

Confusion Matrix
In what follows, the plot of confusion matrix as well as that of accuracy and loss measured on test set for all the models are provided.

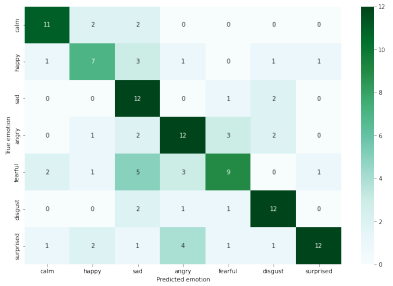


Figure 20: Confusion Matrix of GRU model

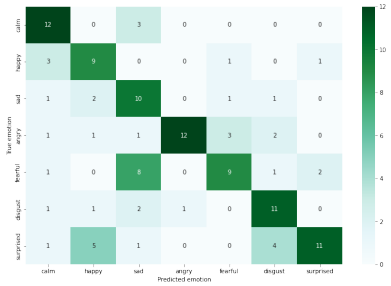


Figure 21: Confusion Matrix of BIGRU model

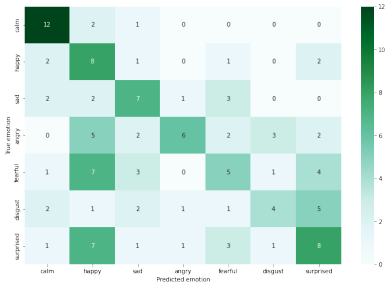


Figure 22: Confusion Matrix of LSTM model

From the confusion matrices we can deducde the following: The most notorious misclassified labels in GRU model were fearful and surprised, which among wrong predictions were mostly classified as sad and angry respectively. The most notorious misclassified labels in BIGRU model were fearful and surprised again. Among wrong predictions, fearful were mostly classified as sad, and surprised were mostly classified as happy or disgust. The most notorious misclassified labels in LSTM model were fearful and surprised, which among wrong predictions were mostly classified as happy and happy respectively.

5 Conclusions and Future Direction

In this project, three recurrent neural network (RNN) models, i.e., long-short term memory (LSTM), gated recurrent unit (GRU) and Bidirectional GRU (BIGRU), were implemented for the task of speech emotion recognition, using mel-frequency cepstral coefficients (MFCCs) of the audio signals. The GRU model outperformed all the other models, and it is significantly more accurate than a random classifier. The limitations and possible improvements of the project are explained in the following: The models used here were only capable of detecting emotions on pre-trained data. However, one can implement it to also receive and classify real-time audios, which are required to follow the same format and audio length of the RAVDESS dataset files. The representation of emotions used in this work were discrete. It is also possible to use a continuous representation and instead of classification, apply regression models to predict the numbers associated with the emotions. Our speculation is that the small size of data is a contributing factor to overfitting, hence either augmenting the data, or even using different datasets while unifying them in the same model probably mitigates the overfitting. Finally, instead of using a single model, one can also use ensemble models. We intend to use an ensemble model of the GRU model with a Convolutional Neural Network (CNN) model from another project that tackles the same task on the same dataset as ours.

References

[1] Shalbbya Ali et al. "Mel Frequency Cepstral Coefficient: A Review". In: EAI, Mar. 2021. DOI: 10.4108/eai.27-2-2020.2303173.

[2] Octavian Cheng, Waleed Abdulla, and Zoran Salcic. "Performance Evaluation of Front-end Processing for Speech Recognition Systems". In: (June 2022).

[3] John Doe. *Cepstral Analysis of Speech*. University of Neverland. 2011. URL: vlab.amrita.edu/index.php?sub=59&brch=164&sim=615&cnt=6.

[4] J. Goodfellow Ian, Bengio Yoshua, and Aaron Courville. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. URL: http://www.deeplearningbook.org/.

[5] Dias Issa, M. Fatih Demirci, and Adnan Yazici. "Speech emotion recognition with deep convolutional neural networks". In: *Biomedical Signal Processing and Control* 59 (2020), p. 101894. ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2020.101894. URL: https://www.sciencedirect.com/science/article/pii/S1746809420300501.

[6] Ruhul Amin Khalil et al. "Speech Emotion Recognition Using Deep Learning Techniques: A Review". In: *IEEE Access* 7 (2019), pp. 117327–117345. DOI: 10.1109/ACCESS.2019.2936124.

[7] Thair Khdour et al. "Arabic Audio News Retrieval System Using Dependent Speaker Mode, Mel Frequency Cepstral Coefficient and Dynamic Time Warping Techniques". In: *Research Journal of Applied Sciences, Engineering and Technology* 7 (June 2014), pp. 5082–5097. DOI: 10.19026/rjaset.7.903.

[8] Peter Knees and Markus Schedl. *Music Similarity and Retrieval*. Vol. 36. Jan. 2016. ISBN: 978-3-662-49720-3. DOI: 10.1007/978-3-662-49722-7.

[9] Michael Kraus. "Voice-only communication enhances empathic accuracy". In: *American Psychologist* 72 (Oct. 2017), pp. 644–654. DOI: 10.1037/amp0000147.

[10] Jinkyu Lee and Ivan Tashev. "High-level Feature Representation using Recurrent Neural Network for Speech Emotion Recognition". In: Sept. 2015. DOI: 10.21437/Interspeech.2015-336.

[11] Steven R. Livingstone and Frank 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". In: *PLOS ONE* 13 (May 2018), pp. 1–35. DOI: 10.1371/journal.pone.0196391. URL: https://doi.org/10.1371/journal.pone.0196391.

[12] Sergej Lugovic, Ivan Dunđer, and Marko Horvat. "Techniques and Applications of Emotion Recognition in Speech". In: May 2016. DOI: 10.1109/MIPRO.2016.7522336.

[13] Brian McFee et al. "librosa: Audio and Music Signal Analysis in Python". In: Jan. 2015, pp. 18–24. DOI: 10.25080/Majora-7b98e3ed-003.

[14] Seyedmahdad Mirsamadi, Emad Barsoum, and Cha Zhang. "Automatic Speech Emotion Recognition Using Recurrent Neural Networks with Local Attention". In: Mar. 2017. DOI: 10.1109/ICASSP.2017.7952552.

[15] Mustaqeem and Soonil Kwon. "Optimal feature selection based speech emotion recognition using two-stream deep convolutional neural network". In: *International Journal of Intelligent Systems* 36.9 (2021), pp. 5116–5135. DOI: https://doi.org/10.1002/int.22505. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/int.22505. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/int.22505.

[16] A.V. Oppenheim and R.W. Schafer. "From frequency to quefrency: a history of the cepstrum". In: *IEEE Signal Processing Magazine* 21.5 (2004), pp. 95–106. DOI: 10.1109/MSP.2004.1328092.

[17] Björn Schuller. "Speech emotion recognition: Two decades in a nutshell, benchmarks, and ongoing trends". In: *Communications of the ACM* 61 (Apr. 2018), pp. 90–99. DOI: 10.1145/3129340.

[18] Jianfeng Zhao, Xia Mao, and Lijiang Chen. "Speech emotion recognition using deep 1D & 2D CNN LSTM networks". In: *Biomedical Signal Processing and Control* 47 (2019), pp. 312–323. ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2018.08.035. URL: https://www.sciencedirect.com/science/article/pii/S1746809418302337.