Speech Emotion Recognition with Librosa

A PROJECT REPORT

In partial fulfilment for the award of the degree
Of
BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Under the guidance of
Dr. Manoj Sachan
Head Of Department (H.O.D)
CSE Department

Submitted by
Pranav Pushpam(1731363)
Vikas Kumar (1731369)

SANT LONGOWAL INSTITUTE OF ENGINEERING AND TECHNOLOGY, LONGOWAL – 148106, DISTRICT- SANGRUR, PUNJAB, INDIA
MAY, 2020

Table of Contents

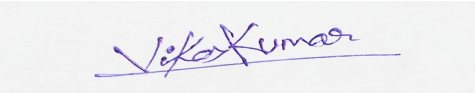
Chapter 3. Problem Definition

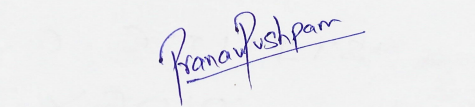
Chapter 4. Objectives

Chapter 5. Methodology Used

CERTIFICATE

This is to certify that the Dissertation Report entitled, "**Speech Emotion Recognition with Librosa**" submitted by "**Pranav Pushpam (GCS-1731363), Vikas Kumar (GCS-1731369)**" to **Sant Longowal Institute of Engineering And Technology,** (Punjab) India, is a record of bonafide Project work carried out by him under my supervision and guidance and is worthy of consideration for the award of the degree of Bachelor Of Engineering in Computer Science & Engineering of the Institute.





Signature of the students

Signature of the Supervisor

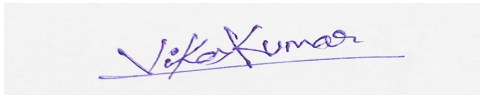
Name of the supervisor : Dr. Manoj Sachan

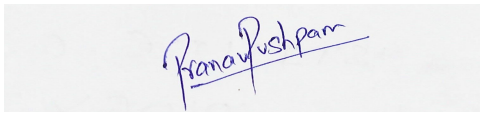
Designation : Head Of Department (HOD)

Date : 20 MAY 2020

DECLARATION

I hereby declare that the project entitled **"Speech Emotion Recognition with Librosa"** submitted for the Bachelor Of Engineering (CSE) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.





Signature of the Student **Date:** 20 MAY 2020

Acknowledgement

If words are considered as a symbol of approval and token of appreciation then let the words play the heralding role expressing my gratitude.

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people whose ceaseless cooperation, constant guidance and encouragement made it possible. We are grateful to our project guide **Dr. Manoj Sachan, Head Of Department, CSE Department** , for the guidance, inspiration and constructive suggestion that helped us in the preparation of this project.

We would like to express our special gratitude and thanks to our mentor **Hutashan Vishal Bhagat, Research Scholar, CSE Department.**

Pranav Pushpam

(GCS-1731363)

Vikas Kumar

(GCS-1731369)

Abstract

Speech is simply the most common method for communicating as people. It is just common at that point only natural then to extend out this correspondence medium to PC applications. We characterize speech emotion recognition (SER) as an assortment of systems that procedure and classify speech signals to detect the embedded emotions.
In simple words, It is the act of attempting to recognize human emotion and affective states from speech. This is the system that will significantly take a shot at the way that voice frequently reflects hidden feelings through tone and pitch. SER is tough because emotions are subjective and annotating audio is challenging. By using this system, we can identify the human emotion like sad, cheerful, calm, angry, happy, fearful, regret, etc. by their speech or voice or we can say using some audio.
In human machine interface application, emotion recognition from the speech signal has been a project topic since many years. To identify the emotions from the speech signal, many systems have been developed.
In this project report speech emotion recognition based on the previous technologies which uses different classifiers for the emotion recognition is reviewed. Speech has several characteristic features such as naturalness and efficiency, which makes it an attractive interface medium. It is possible to express emotions and attitudes through speech.
Here in this project, study has been carried out to recognize human emotion through speech using the MFCCs. To recognize emotion through speech various speech features were extracted. Based on these speech features Classification of the emotions has been done and the classification performance of Convolutional Neural Network is discussed. Here emotion recognition is done for different emotions like neutral, happy, sad, boredom anger, and fear. The classification performance is based on extracted features. Inference about the performance and limitation of speech emotion recognition systems based on the different classifiers are also discussed.
Mel-frequency cepstrum coefficients (MFCC) and modulation spectral (MS) features are extracted from the speech signals and used to train different classifiers.

LIST OF TABLES

LIST OF FIGURES

Chapter 1. Introduction

Project Overview/Specifications

The idea behind creating this project was to build a machine learning model that could detect emotions from the speech we have with each other all the time. Nowadays personalization is something that is needed in all the things we experience everyday.
Detecting emotions is one of the most important marketing strategies in today's world. You could personalize different things for an individual specifically to suit their interest. For this reason, we decided to do a project where we could detect a person's emotions just by their voice which will let us manage many AI related applications. Some examples could be including call centers to play music when one is angry on the call. Another could be a smart car slowing down when one is angry or fearful. As a result this type of application has much potential in the world that would benefit companies and also even safety to consumers.

In this project, I will utilize the libraries librosa, sound record, and sklearn to construct a model .We would use MFCCs to be our input feature. This will have the option to perceive feelings from sound documents. At that point, we'll instate the CNN model with Keras and construct it with 7 layers — 6 Conv1D layers followed by a Dense layer and train the model.

1. This project requires some information on points like **Python, sklearn, librosa, and so on.**
2. **Python-**Python is easy to learn and work on with the language. It is an elevated level, broadly useful programming and profoundly intruded on language.
3. **Librosa** -Librosa is a Python library for examining sound and music. It has a compliment bundle format, institutionalized interfaces, and names, in reverse similarity, measured capacities, and meaningful/readable code.
4. Point to be noted, "Programmers can explore different avenues regarding the programming language as demonstrated by their comfort level and data" and can change the recently referenced language according to them.
5. **CNN:** Using Convolutional Neural Network to recognize emotion from the audio recording.

Software Requirement -

- Programming language - Python
- ANACONDA 3-64bit
- Jupyter Notebook
- Operating System - any os like a window, ubuntu.

Kit required to develop Speech Emotion Recognition using Python:

- No kit required

Technologies you will learn by working on Speech Emotion Recognition using Python:

- Python

Analyzing audio signals

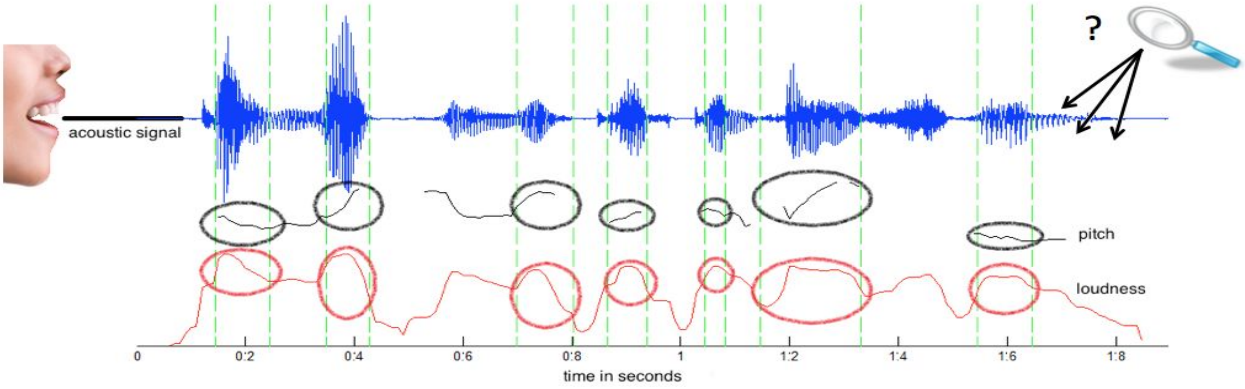


Fig: 1.1

Fig: 1.2

Aim Of The Project:

The proposed project aimed to:

- develop and test new efficient feature extraction methods for an automatic recognition and classification of stress and emotion in speech;
- determine if the recent laryngological studies indicating a nonlinear character of speech production can be used to derive efficient feature extraction methods for an automatic recognition and classification of stress and emotion in speech.

Package Required

glob2==0.6, ipython==7.4.0p, keras==2.3.1, librosa==0.6.3, matplotlib==3.0.2, numpy==1.16.2, pandas==0.25.1, plotly==4.6.0, scipy==1.2.1, seaborn==0.9.0, sklearn==0.20.1, tensorflow==1.13.1, tqdm==4.29.0

Data Description:

Data Used: We got audio datasets with around 2000 audio files which were in the wav format from the following websites:

1: http://neuron.arts.ryerson.ca/ravdess/?f=3,
2: http://kahlan.eps.surrey.ac.uk/savee/Download.html

The first website contains speech data which is available in three different formats.
1. Audio Visual – Video with speech
2. Speech – Audio only
3. Visual – Video only

Data Set: The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)

- 12 Actors & 12 Actresses recorded speech and song versions respectively.
- Actor no.18 does not have song version data.
- Emotion Disgust, Neutral and Surprised are not included in the song version data.

We went with the Audio only zip file because we are dealing with finding emotions from speech. The zip file consisted of around 1500 audio files which were in wav format.

The second website contains around 500 audio speeches from four different actors with different emotions.
The next step involves organizing the audio files. Each audio file has a unique identifier at the 6^{th} position of the file name which can be used to determine the emotion the audio file consists of. We have 5 different emotions in our dataset.

1. Calm 2. Happy 3. Sad 4. Angry 5. Fearful

- We used Librosa library in Python to process and extract features from the audio files. Librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems. Using the librosa library we were able to extract features i.e MFCC(Mel Frequency Cepstral Coefficient). MFCCs are a feature widely used in automatic speech and speaker recognition. We also separated out the females and males voices by using the identifiers provided in the website. This was because as an experiment we found out that separating male and female voices increased by 15%. It could be because the pitch of the voice was affecting the results.
- Each audio file gave us many features which were basically an array of many values. These features were then appended by the labels which we created in the previous step.
- The next step involved dealing with the missing features for some audio files which were shorter in length. We increased the sampling rate by twice to get the unique features of each emotional speech. We didn't increase the sampling frequency even more since it might collect noise thus affecting the results.

Chapter 2: Literature review

The importance of emotion recognition is getting popular with improving user experience and the engagement of Voice User Interfaces (VUI's). Developing emotion recognition systems that are based on speech has practical application benefits. However, these benefits are somewhat negated by the real-world background noise impairing speech-based emotion recognition performance when the system is employed in practical applications.

Speech Emotion Recognition (SER) is one of the most challenging tasks in the speech signal analysis domain, it is a research area problem which tries to infer the emotion from the speech signals.

Where can we use it ?

Even though it isn't that popular, SER has entered so many areas these years, including:

- **The medical field:** In the world of telemedicine where patients are evaluated over mobile platforms, the ability for a medical professional to discern what the patient is actually feeling can be useful in the healing process.
- **Customer service:** In call center conversation may be used to analyze behavioral study of call attendants with the customers which helps to improve the quality of service.
- **Recommender systems:** Can be useful to recommend products to customers based on their emotion towards that product.

I. INTRODUCTION

In today's digital era, speech signals become a mode of communication between humans and machines which is possible by various technological advancements. Speech recognition techniques with methodologies and signal processing techniques have led to Speech-to-Text (STT) technology which uses mobile phones as a mode of communication. Speech Recognition

is the fastest growing research topic in which attempts to recognize speech signals. This leads to Speech Emotion Recognition(SER) growing research topic in which lots of advancements can lead to advancements in various fields like automatic translation systems, machine to human interaction, used in synthesizing speech from text so on. In contrast the project focuses on survey and review of various speech extraction features, emotional speech databases, classifier algorithms and so on. Problems present in various topics were addressed.

This project is organized as follows :

Section 2 describes background information about speech recognition, emotion recognition system, applications of emotion recognition.

Section 3 explains the methods of feature extraction and optimization from speech signals.

Section 4 compares various speech emotional databases prepared .

Section 5 contains various classifier algorithms for classifying speech signals according to the emotion inferred.

Finally, a conclusion is given in **section 6.**

II. BACKGROUND INFORMATION

A. SPEECH RECOGNITION

Speech Recognition is the technology that deals with techniques and methodologies to recognize the speech from the speech signals. Various technological advancements in the field of artificial intelligence and signal processing techniques, recognition of emotion made it easier and possible. It is also known as „Automatic Speech Recognition". It is found that voice can be the next medium for communicating with machines especially when computer-based systems. A Need for inferring emotion from spoken utterances increases exponentially. Since there is an enormous development in the field of Voice Recognition. There are many voice products developed like Amazon Alex, Google Home, Apple HomePod which functions mainly on voice based commands. It is evident that Voice will be the better medium for communicating to the machines.

B. EMOTION RECOGNITION

Basically, Emotion Recognition deals with the study of inferring emotions, methods used for inferring. Emotions can be recognized from facial expressions, speech signals. Various

techniques have been developed to find the emotions such as signal processing, machine learning, neural networks, computer vision. Emotion analysis, Emotion Recognition are being studied and developed all over the world. Emotion Recognition is gaining its popularity in research which is the key to solving many problems also makes life easier. The main need of Emotion Recognition from Speech is challenging tasks in Artificial Intelligence where speech signals are alone an input for the computer systems. Speech Emotion Recognition (SER) is also used in various fields like BPO Centre and Call Centre to detect the emotion useful for identifying the happiness of the customer about the product, IVR Systems to enhance the speech interaction, to solve various language ambiguities and adaption of computer systems according to the mood and emotion of an individual.

C. SPEECH EMOTION RECOGNITION

Speech Emotion Recognition is a research area problem which tries to infer the emotion from the speech signals. Various surveys state that advancement in emotion detection will make a lot of systems easier and hence make the world a better place to live. SER has its own application which is explained later. Emotion Recognition is the challenging problem in ways such as emotion may differ based on the environment, culture, individual face reaction leads to ambiguous findings; speech corpus is not enough to accurately infer the emotion; lack of speech database in many languages. Survey on speech emotion recognition which helps a lot in exploring speech emotion recognition

D. DEEP LEARNING

Deep Learning is machine learning techniques in which data models are designed bound to a specific task. Deep learning in neural networks is used for various tasks such image recognition, classification tasks, decision making, pattern recognition etc. Various other Deep Learning techniques such as multimodal deep learning used for feature extraction, image recognition made at ease.

E. APPLICATIONS OF EMOTION RECOGNITION

Emotion Recognition is used in call centers for classifying calls according to emotions. Emotion Recognition serves as the performance parameter for conversational analysis thus identifying the unsatisfied customer, customer satisfaction and so on. SER is used in-car board systems based on information of the mental state of the driver can be provided to the system to initiate his/her safety preventing accidents to happen.

III. LITERATURE SURVEY

Complete review on speech emotion recognition is explained in which reviews properties of dataset, speech emotion recognition study classifier choice. Various acoustic features of speech are investigated and some of the classifier methods are analyzed which is helpful in the further investigation of modern methods of emotion recognition. This project investigated the prediction of the next reactions from emotional vocal signals based on the recognition of emotions, using different categories of classifiers. Some of the classification algorithms like K-NN, Random Forest are used to classify emotion accordingly. Recurrent Neural networks arise enormously which tries to solve many problems in the field of data science. Deep RNN like LSTM, Bi-directional LSTM trained for acoustic features are used in. Various ranges of CNN are being implemented and trained for speech emotion recognition are evaluated in. Emotion is inferred from speech signals using filter banks and Deep CNN which shows high accuracy rate which gives an inference that deep learning can also be used for emotion detection. Speech emotion recognition can be also performed using image spectrograms with deep convolutional networks which is implemented in.

IV. PROPOSED METHODOLOGY

This section explains the proposed methodology, emotion database used for project, Inception model.

A. EMOTION DATABASE

Data Used: We got audio datasets with around 2000 audio files which were in the wav format from the following websites:
1: http://neuron.arts.ryerson.ca/ravdess/?f=3,
2: http://kahlan.eps.surrey.ac.uk/savee/Download.html
The first website contains speech data which is available in three different formats.
1. Audio Visual – Video with speech
2. Speech – Audio only
3. Visual – Video only

Data Set: The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)

- 12 Actors & 12 Actresses recorded speech and song versions respectively.
- Actor no.18 does not have song version data.
- Emotion Disgust, Neutral and Surprised are not included in the song version data.

We went with the Audio only zip file because we are dealing with finding emotions from speech. The zip file consisted of around 1500 audio files which were in wav format.

The second website contains around 500 audio speeches from four different actors with different emotions.
The next step involves organizing the audio files. Each audio file has a unique identifier at the 6th position of the file name which can be used to determine the emotion the audio file consists of. We have 5 different emotions in our dataset.

1. Calm 2. Happy 3. Sad 4. Angry 5. Fearful

B. TRANSFER LEARNING

Transfer learning is one of the machine learning models which uses the knowledge gained from solving one problem and is incorporated to solve another problem. It is evident that Transfer learning solves many problems within a short interval of time. Transfer Learning is incorporated whenever there is any need to reduce computation cost, achieve accuracy with less training i.e; Inception Net Architecture

C. INCEPTION NET V3 MODEL

Inception Net v3 Model is used to build an emotion recognition model. Inception evolved from GoogLeNet Architecture with some enhancements. Inception model is used for automatic image classification and image labelling according to the image. Inception-v3 is used for image classification in Google Image Search. Inception-v3 achieved a top 5.6% error rate in ILSVRC 2012 classification challenge validation. Inception Net model which consists of an Inception module which concatenates all the output of 1x1,3x3,5x5 filters. Inception net consists of a network in a network which consist of three inception modules that are embedded inside the inception architecture which helps in reduction of numerical array.

D. PREPARATION OF TRAINING DATASET

All the audio clips from the RAVDESS databases are pulled out from various sessions. Using the emotion evaluation report which is given along with the database, various wav files are labeled and categorized into seven ranges of emotions as mentioned earlier. Speech signals in .wav format are converted into spectrogram images within the emotion class.

V. EXPERIMENTAL SETUP

This section contains explanations about experimental setup, libraries used for Speech Emotion Recognition which helps in emotion recognition.

A. SYSTEM SETUP

For performing the experiment I"ve used a system setup consisting of Core i3 6th Generation 2.00 GHz Processor, 4GB RAM, System type: 64-bit operating system. For deep learning I"ve used TensorFlow 1.5 for implementing the Inception net model and Tensor Board for visualizing the learning, graphs, histograms and so on.

B. TRAINING METHOD

All images labeled with respective emotions are prepared for training the model. The proposed CNN model was implemented using TensorFlow. The spectrogram images generated from the RAVDESS are resized . Spectrograms were generated from all the audio files in the dataset. For each emotion, an Image range of about 500 for each class of emotion is collected from the database. The training process was run for 700 epochs with a batch size set to 16. The training took around 24hrs and the best accuracy was achieved after 700epochs. On the training set, a loss of 61.25% was achieved, whereas 65.22% loss was recorded on the test set. An accuracy of 75.21 % was achieved per spectrogram.

VI. RESULT & ANALYSIS

An accuracy rate of about 75.21% is achieved from the data model for predicting the emotions.

Chapter 3 : Problem Definition

Steps include:

- Importing the required libraries.
- Reading the data.
- Plotting the audio file's waveform and its spectrogram.
- Defining the truth label.
- Data Splitting.
- Getting the features of audio files using librosa.
- Data Augmentation.
- Changing dimension for CNN model.
- Removed the whole training part for avoiding unnecessary long epochs list.
    - Saving the model
    - Loading the model
- Predicting emotions on the test data.
    - Actual v/s Predicted emotions

Major Obstacles:

- Emotions are subjective, people would interpret it differently. It is hard to define the notion of emotions.
- Annotating an audio recording is challenging. Should we label a single word, sentence or a whole conversation? How many emotions should we define to recognize?
- Collecting data is complex. There is lots of audio data that can be achieved from films or news. However, both of them are biased since news reporting has to be neutral and actors' emotions are imitated. It is hard to look for neutral audio recording without any bias.
- Labeling data requires high human and time cost. Unlike drawing a bounding box on an image, it requires trained personnel to listen to the whole audio recording, analyze it and give an annotation. The annotation result has to be evaluated by multiple individuals due to its subjectivity.

Problem statements

- In recent years, speech scientists and engineers, who had tended to disregard pragmatic and paralinguistic aspects of speech in their effort to develop models of speech communication for speech technology applications, have started to devote more attention to speaker attitudes and emotions.

- often in the interest to increase the acceptability of speech technology for human users . The speech signal communicates linguistic information between speakers as well as paralinguistic information about speaker's emotions, personalities, attitudes, feelings, levels of stress and current mental states.
- Just as effective human-to-human communication is virtually impossible without speakers being able to detect and understand each other's emotions, human-machine communication suffers from significant inefficiencies because machines cannot understand our emotions or generate emotional responses.
- Words are not enough to correctly understand the mood and intention of a speaker and thus the introduction of human social skills to human-machine communication is of paramount importance. This can be achieved by the researching and creating methods of speech modeling and analysis that embrace the signal, linguistic and emotional aspects of communication.
- Speech emotion recognition and classification is aimed to automatically detect emotions in speech signals through analyzing the vocal behavior as a marker of affect (e.g. emotions, moods and stresses), focusing on the nonverbal aspect of speech. The assumption is that most affective states involve physiological reactions, like the changes in the autonomic and somatic nervous systems, which in turn influences the speech production process, and causes the changes of several voice parameters . The automatic affect recognition in speech has applications in behavioral and mental health science, human to machine communication, social robotics, security systems, computer games, online education, commercial field and medicine.
- It can be used to improve the robustness of speech and speaker recognition systems . Moreover, by assessing a speaker's speech, emotion classification can support automatic assessment of mental states of people working in dangerous environments (e.g. chemicals, explosives) and people undertaking high levels of responsibility (e.g. pilots, surgeons).
- Various clinical applications of affect analysis in speech have been reported in diagnosis of conditions such as depression , autism , Alzheimer and dementia , and schizophrenia .
- These works demonstrate the importance of speech as a diagnostic signal providing valid information about a speaker's mental and physiological state.
- Call systems can use emotion recognition to sort emergency telephone messages, or cope with disputes through monitoring the mental states (levels of satisfaction) of customers. Another commercial application of emotion detection system is the interactive game industry offering the sensation of naturalistic human-like interaction, and the entertainment industry facilitating the intelligent agents with sensitivity to player's mood as well as the ability to respond accordingly through affective voice or face expression.

Key Takeaway

- Emotions are subjective and it is hard to notate them.
- We should define the emotions that are suitable for our own project objective.
- Do not always trust the content from GitHub even if it has lots of stars.
- Be aware of the data splitting.
- Exploratory Data Analysis always grants us good insight, and you have to be patient when you work on audio data!
- Deciding the input for your model: a sentence, a recording or an utterance?
- Lack of data is a crucial factor to achieve success in SER, however, it is complex and very expensive to build a good speech emotion dataset.
- Simplified your model when you lack data

Chapter 4. Objectives

Speech Emotion Recognition

- To build a model to recognize emotion from speech using the librosa and sklearn libraries and the RAVDESS dataset.
- In this Python project, we will use the libraries librosa, sound file, and sklearn (among others) to build a model using an MLPClassifier. This will be able to recognize emotion from sound files. We will load the data, extract features from it, then split the dataset into training and testing sets. Then, we'll initialize an MLPClassifier and train the model. Finally, we'll calculate the accuracy of our model.

The Dataset

For this Python project, we'll use the RAVDESS dataset; this is the Ryerson Audio-Visual Database of Emotional Speech and Song dataset, and is free to download. This dataset has 7356 files rated by 247 individuals 10 times on emotional validity, intensity, and genuineness. The entire dataset is 24.8GB from 24 actors, but we've lowered the sample rate on all the files, and you can download it here.

Prerequisites

You'll need to install the following libraries with pip:

```
1. pip install librosa soundfile numpy sklearn pyaudio
```

Fig: 4.1

Chapter 5. Methodology Used

In this work, we conduct an extensive comparison of various approaches to speech based emotion recognition systems. The analyses were carried out on audio recordings from **Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)**. After pre-processing the raw audio files, features such as **Log-Mel Spectrogram, Mel-Frequency Cepstral Coefficients (MFCCs), pitch and energy** were considered. The significance of these features for emotion classification was compared by applying methods such as **Long Short Term Memory (LSTM)**, **Convolutional Neural Networks (CNNs), Hidden Markov Models (HMMs) and Deep Neural Networks (DNNs).** On the 14-class (2 genders x 7 emotions) classification task, an **accuracy of 75.21% was achieved with a 4-layer 2 dimensional CNN using the Log-Mel Spectrogram features.** We also observe that, in emotion recognition, the choice of audio features impacts the results much more than the model complexity.

Fig: 5.1

Using Convolutional Neural Network to recognize emotion from the audio recording.

Data Set:

The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)

- 12 Actors & 12 Actresses recorded speech and song version respectively.
- Actor no.18 does not have song version data.
- Emotion Disgust, Neutral and Surprised are not included in the song version data.

Total Class:

Emotion	Speech Sample Count	Song Sample Count	Summed Count
Neutral	96	92	188
Calm	192	184	376
Happy	192	184	376
Sad	192	184	376
Angry	192	184	376
Fearful	192	184	376
Disgust	192	0	192
Surprised	192	0	192
Total	1440	1012	2452

Fig: 5.2

Here is the emotion class distribution bar chart.

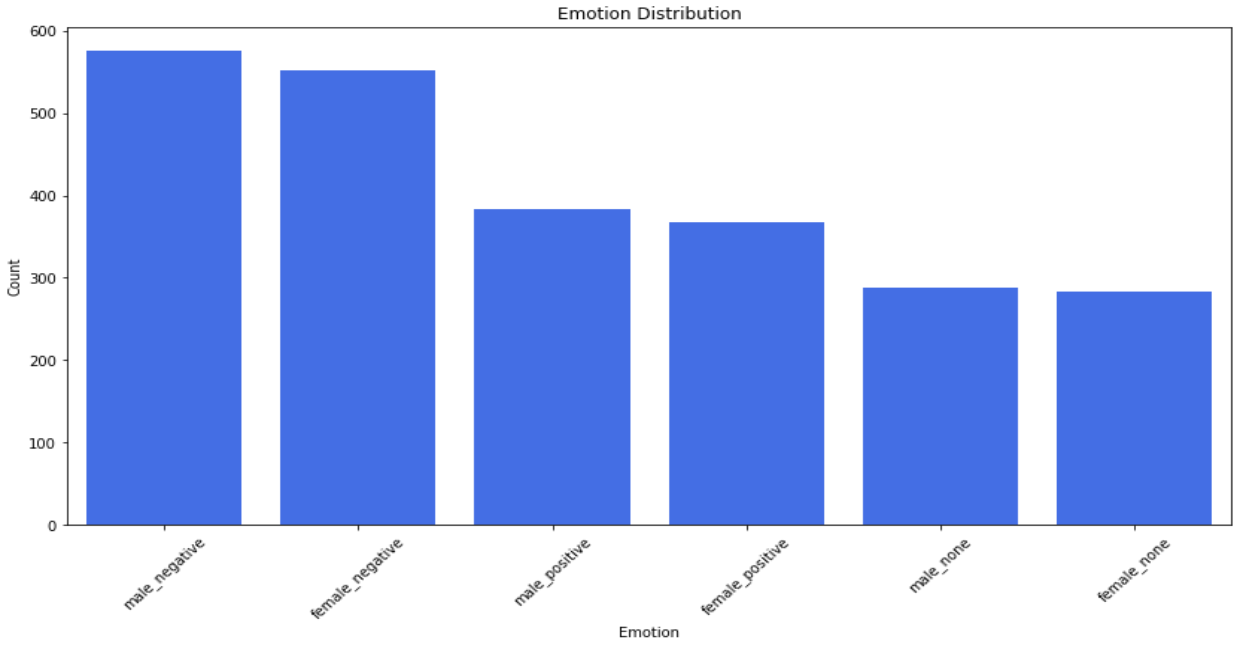
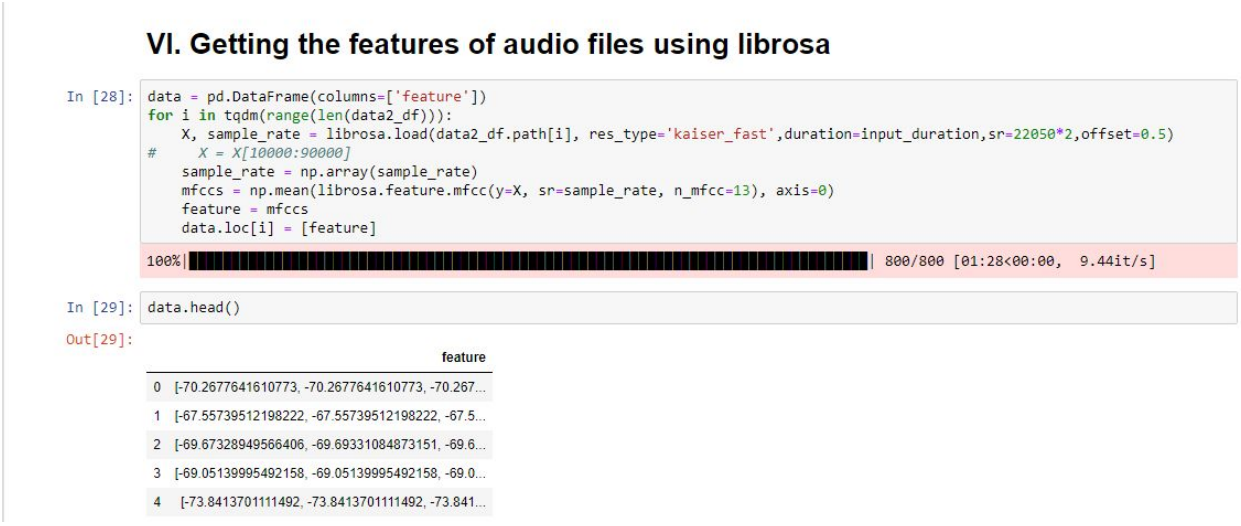


Fig: 5.3

Feature Extraction:

The next step involves extracting the features from the audio files which will help our model learn between these audio files. For feature extraction we make use of the LibROSA library in python which is one of the libraries used for audio analysis.
When we do Speech Recognition tasks, MFCCs are the state-of-the-art feature since it was invented in the 1980s.
This shape determines what sound comes out. If we can determine the shape accurately, this should give us an accurate representation of the phoneme being produced. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.



VI. Getting the features of audio files using librosa

In [28]:
```
data = pd.DataFrame(columns=['feature'])
for i in tqdm(range(len(data2_df))):
    X, sample_rate = librosa.load(data2_df.path[i], res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
#     X = X[10000:90000]
    sample_rate = np.array(sample_rate)
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
    feature = mfccs
    data.loc[i] = [feature]
```

```
100%|██████████| 800/800 [01:28<00:00,  9.44it/s]
```

In [29]:
```
data.head()
```

Out[29]:

	feature
0	[-70.2677641610773, -70.2677641610773, -70.267...
1	[-67.55739512198222, -67.55739512198222, -67.5...
2	[-69.67328949566406, -69.69331084873151, -69.6...
3	[-69.05139995492158, -69.05139995492158, -69.0...
4	[-73.8413701111492, -73.8413701111492, -73.841...

Fig: 5.4

- The sampling rate of each file is doubled keeping sampling frequency constant to get more features which will help classify the audio file when the size of the dataset is small.

Waveform and Spectrogram

We tested out one of the audio files to know its features by plotting its waveform and spectrogram.

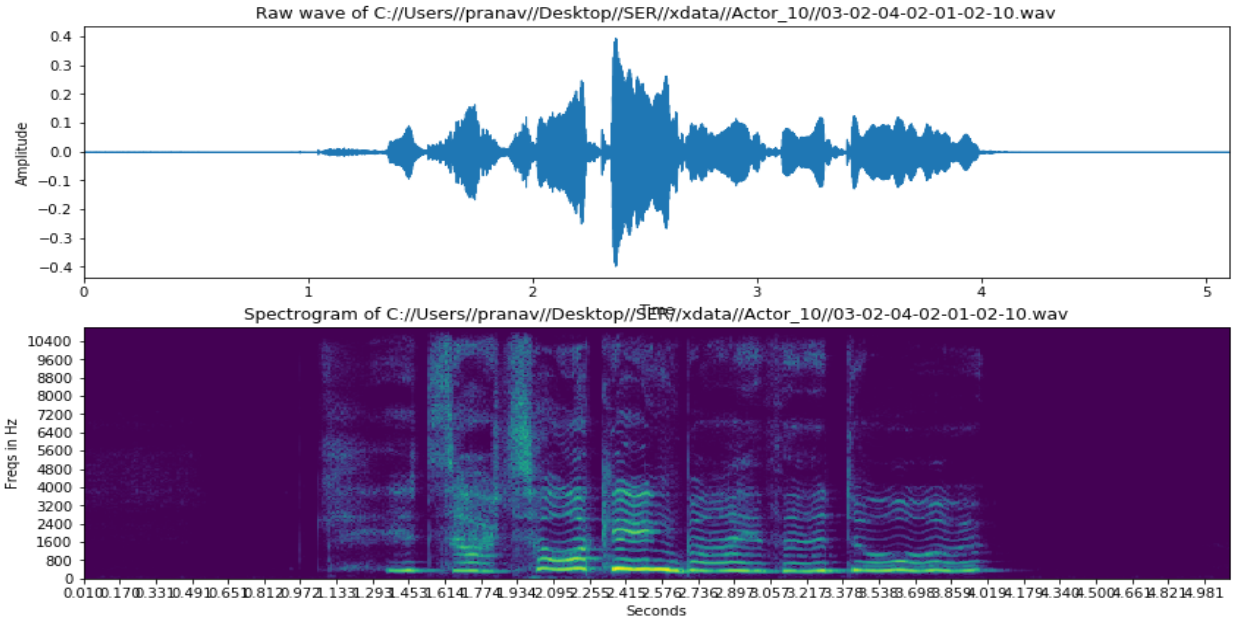


Fig: 5.5

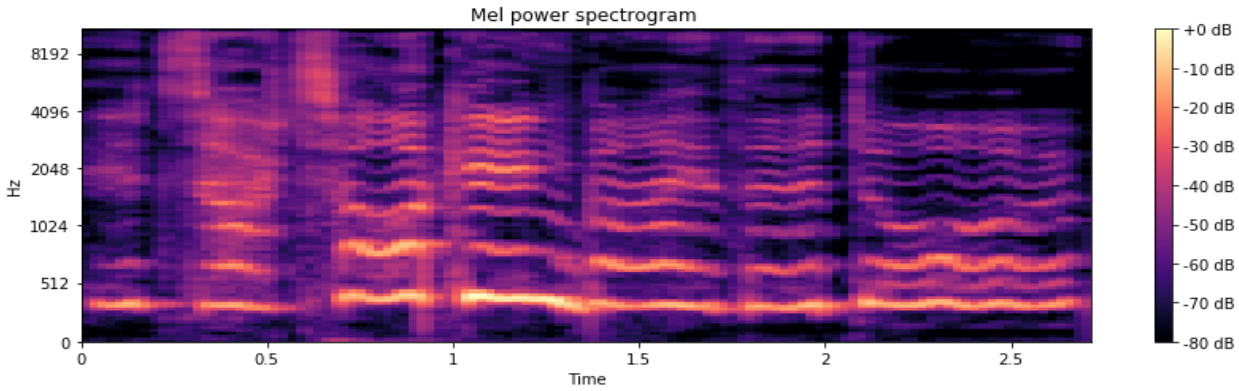


Fig: 5.6

We would use MFCCs to be our input feature. If you want a thorough understanding of MFCCs. Loading audio data and converting it to MFCCs format can be easily done by the Python package librosa.

Default Model Architecture:

We developed the CNN model with Keras and constructed with 7 layers — 6 Conv1D layers followed by a Dense layer.

Original model

```
In [59]: # Original Model

         # model = Sequential()
         # model.add(Conv1D(256, 5,padding='same', input_shape=(X_train.shape[1],1)))
         # model.add(Activation('relu'))
         # model.add(Conv1D(128, 5,padding='same'))
         # model.add(Activation('relu'))
         # model.add(Dropout(0.1))
         # model.add(MaxPooling1D(pool_size=(8)))
         # model.add(Conv1D(128, 5,padding='same',))
         # model.add(Activation('relu'))
         # model.add(Conv1D(128, 5,padding='same',))
         # model.add(Activation('relu'))
         # model.add(Conv1D(128, 5,padding='same',))
         # model.add(Activation('relu'))
         # model.add(Dropout(0.2))
         # model.add(Conv1D(128, 5,padding='same',))
         # model.add(Activation('relu'))
         # model.add(Flatten())
         # model.add(Dense(5))
         # model.add(Activation('softmax'))
         # opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
```

Fig: 5.7

New model

```
In [58]: # New model
         model = Sequential()
         model.add(Conv1D(256, 8, padding='same',input_shape=(X_train.shape[1],1)))
         model.add(Activation('relu'))
         model.add(Conv1D(256, 8, padding='same'))
         model.add(BatchNormalization())
         model.add(Activation('relu'))
         model.add(Dropout(0.25))
         model.add(MaxPooling1D(pool_size=(8)))
         model.add(Conv1D(128, 8, padding='same'))
         model.add(Activation('relu'))
         model.add(Conv1D(128, 8, padding='same'))
         model.add(Activation('relu'))
         model.add(Conv1D(128, 8, padding='same'))
         model.add(Activation('relu'))
         model.add(Conv1D(128, 8, padding='same'))
         model.add(BatchNormalization())
         model.add(Activation('relu'))
         model.add(Dropout(0.25))
         model.add(MaxPooling1D(pool_size=(8)))
         model.add(Conv1D(64, 8, padding='same'))
         model.add(Activation('relu'))
         model.add(Conv1D(64, 8, padding='same'))
         model.add(Activation('relu'))
         model.add(Flatten())


         # Edit according to target class no.
         model.add(Dense(2))
         model.add(Activation('softmax'))
         opt = keras.optimizers.SGD(lr=0.0001, momentum=0.0, decay=0.0, nesterov=False)
```

Fig: 5.8

layer #4 and #5 in the notebook ,the model weight file does not fit the network provided, thus, I cannot load the weight provided and replicate its **result 75.21% Testing Accuracy**.

The model only simply trained with **batch_size=16 and 700 epochs** without any learning rate schedule, etc.

Compile Model

Fig: 5.9

Fit Model

Cnnhistory = model.fit(x_traincnn, y_train, batch_size=16, epochs=700,
Validation_data = (x_testcnn, y_test), callbacks = [mcp_save, lr_reduce])

Its loss function is categorical_crossentropy and the evaluation metric is accuracy.

My Experiment

Exploratory Data Analysis:

In the RADVESS dataset, each actor has to perform 8 emotions by saying and singing two sentences and two times for each. As a result, each actor would induce 4 samples for each emotion except neutral, disgust and surprised since there is no singing data for these emotions. Each audio wave is around 4 second, the first and last second are most likely silenced.

The standard sentences are:

1. Kids are talking by the door.
2. Dogs are sitting by the door.

Observation:

After I selected 1 actor and 1 actress's dataset and listened to all of them. I found out male and female are expressing their emotions in a different way. Here are some findings:

- Male's Angry is simply increased in volume.
- Male's Happy and Sad significant features were laughing and crying tones in the silenced period in the audio.
- Female's Happy, Angry and Sad are increased in volume.
- Female's Disgust would add vomiting sound inside.

Replicating Result:

As they excluded the class neutral, disgusted and surprised to do a 10 class recognition for the RAVDESS dataset.

I tried to replicate his result with the model provided, I can achieve a result of

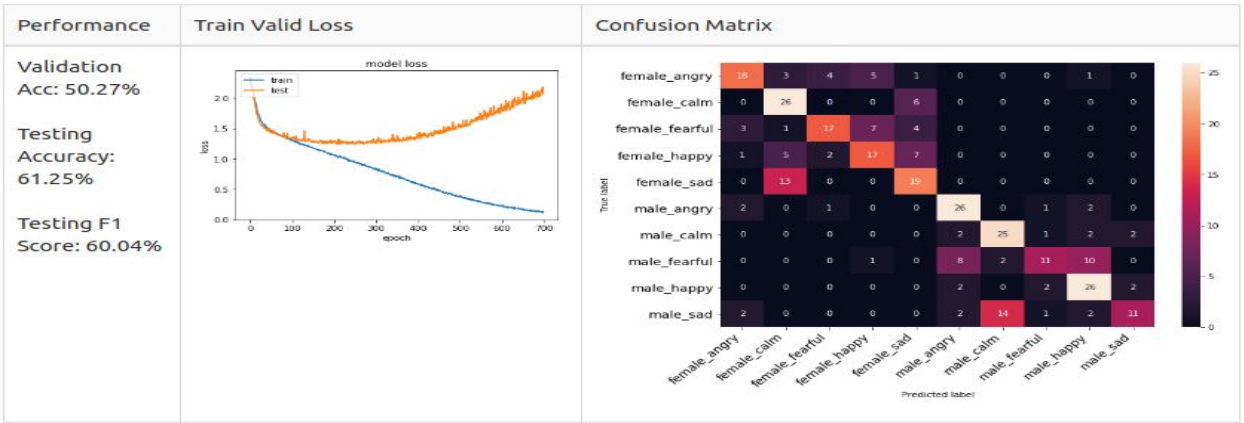


Fig: 5.10

However, I found out there is a data leakage problem where the validation set used in the training phase is identical to the test set. So, I re-do the data splitting part by isolating two actors and two actresses data into the test set which makes sure it is unseen in the training phase.

- Actor no. 1–20 are used for Train / Valid sets with 8:2 splitting ratio.
- Actor no. 21–24 are isolated for testing usage.
- Train Set Shape: (1248, 216, 1)
- Valid Set Shape: (312, 216, 1)
- Test Set Shape: (320, 216, 1) — (Isolated)
-

I re-trained the model with the new data-splitting setting and here is the result:

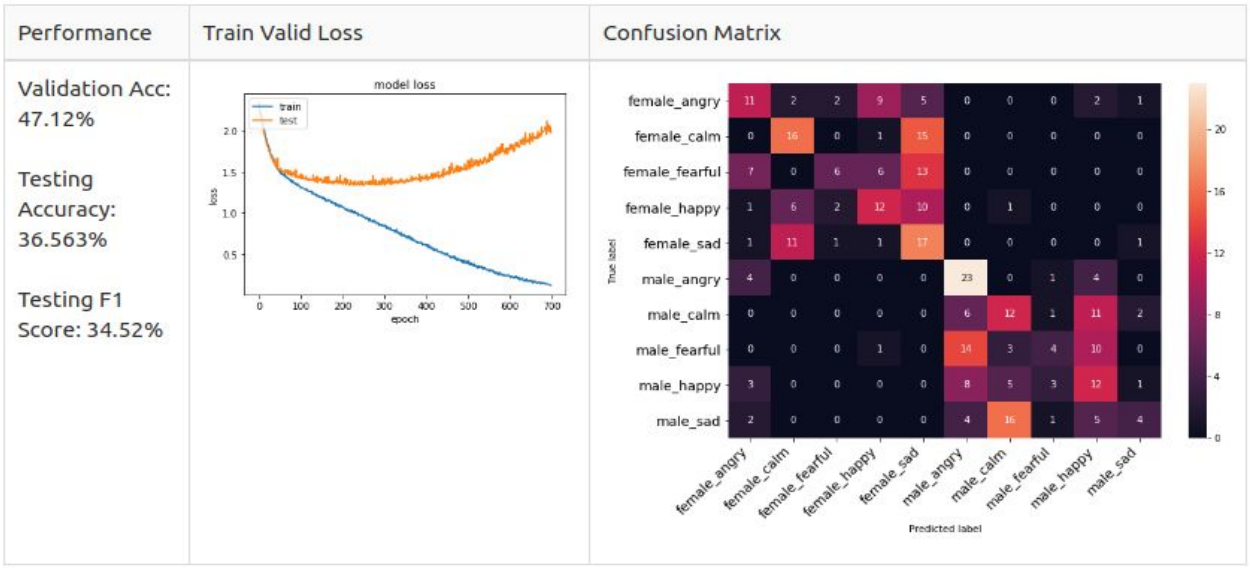


Fig: 5.11

Benchmark:

From the train valid loss graph, we can see the model cannot even converge well with 10 target classes. Thus, I decided to reduce the complexity of my model by recognizing male emotions only. I isolated the two actors to be the test set, and the rest would be the train/valid set with 8:2 Stratified Shuffle Split which ensures there is no class imbalance in the dataset. Afterward, I trained both male and female data separately to explore the benchmark.

Male Dataset

- Train Set = 640 samples from actor 1- 10.
- Valid Set = 160 samples from actor 1- 10.
- Test Set = 160 samples from actor 11- 12.

Male Baseline

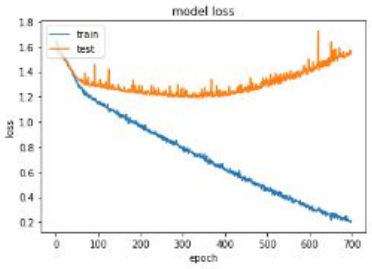
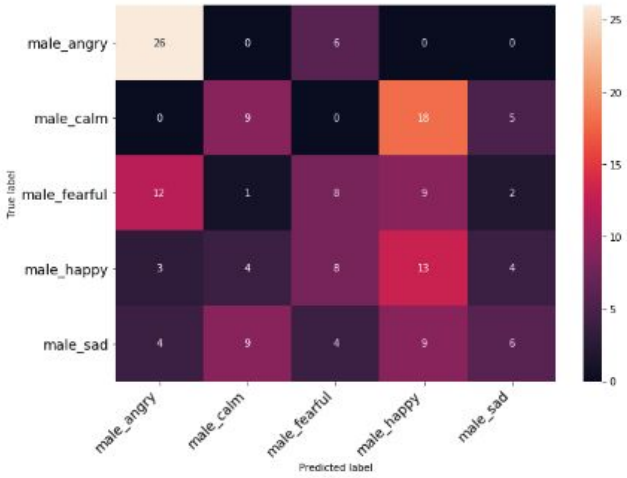
Performance	Train Valid Loss	Confusion Matrix
Validation Acc: 56.25% Testing Accuracy: 38.75% Testing F1 Score: 36.89%		

Fig: 5.12

Female Dataset

- Train Set = 608 samples from actress 1- 10.
- Valid Set = 152 samples from actress 1- 10.
- Test Set = 160 samples from actress 11- 12.

Female Baseline

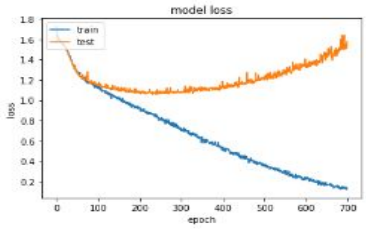
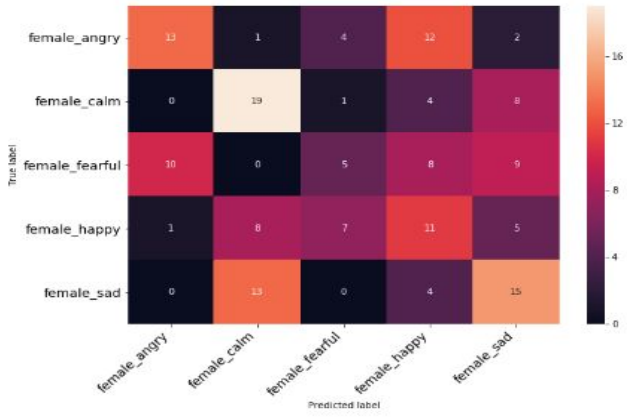
Performance	Train Valid Loss	Confusion Matrix
Validation Acc: 52.63% Testing Accuracy: 39.375% Testing F1 Score: 38.426%		

Fig: 5.13

As you can see, the confusion matrix of the male and female model is different.

- Male: Angry and Happy are the dominant predicted classes in the male model but they are unlikely to mix up.

- Female: Sad and Happy are the dominant predicted classes in the female model and Angry and Happy are very likely to mix up.

- Referring to the observation form the EDA section, I suspect the reason for female Angry and Happy are very likely to mix up is because their expression method is simply increasing the volume of the speech.
- On top of it, I wonder what if I further simplify the model by reducing the target class to Positive, Neutral and Negative or even Positive and Negative only. So, I grouped the emotions into 2 class and 3 class.

2 Class:

- Positive: happy, calm.
- Negative: angry, fearful, sad.

3 Class:

- Positive: happy.
- Neutral: calm, neutral.
- Negative: angry, fearful, sad.

(Added neutral to the 3 class to explore the result.)

Before I do the training , I tune the model architecture with the male data by doing 5 class recognition.

Set the target class number

```
In [58]: # New model
model = Sequential()
model.add(Conv1D(256, 8, padding='same',input_shape=(X_train.shape[1],1)))
model.add(Activation('relu'))
model.add(Conv1D(256, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Flatten())


# Edit according to target class no.
model.add(Dense(2))
model.add(Activation('softmax'))
opt = keras.optimizers.SGD(lr=0.0001, momentum=0.0, decay=0.0, nesterov=False)
```

Fig: 5.14

target_class = 5

I added 2 Conv1D layers, 1 MaxPooling1D layer and 2 Batch Normalization layers, moreover, I changed the dropout value to 0.25. Lastly, I changed the optimizer to SGD with 0.0001 learning rate.

lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=20, min_lr=0.000001)
mcp_save = ModelCheckpoint('model/baseline_2class_np.h5', save_best_only=True, monitor='val_loss', mode='min')
cnnhistory=model.fit(x_traincnn, y_train, batch_size=16, epochs=700, validation_data=(x_testcnn, y_test), callbacks=[mcp_save, lr_reduce])

IX. Removed the whole training part for avoiding unnecessary long epochs list

```
In [*]: # Model Training

lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=20, min_lr=0.000001)
# set the model name accordingly.
mcp_save = ModelCheckpoint('C://Users//pranav//Desktop//SER//saved_models//Emotion_Voice_Detection_Model.h5', save_best_only=True
cnnhistory=model.fit(x_traincnn, y_train, batch_size=16, epochs=700,
                     validation_data=(x_testcnn, y_test), callbacks=[mcp_save, lr_reduce])

Train on 1920 samples, validate on 480 samples
Epoch 1/700
1920/1920 [==============================] - ETA: 1:06 - loss: 0.6422 - accuracy: 0.6250 - fscore: 0.625 - ETA: 1:04 - loss: 0.
6326 - accuracy: 0.6875 - fscore: 0.687 - ETA: 1:03 - loss: 0.6491 - accuracy: 0.6458 - fscore: 0.645 - ETA: 1:04 - loss: 0.645
2 - accuracy: 0.6719 - fscore: 0.671 - ETA: 1:03 - loss: 0.6432 - accuracy: 0.6625 - fscore: 0.662 - ETA: 1:03 - loss: 0.6540 -
accuracy: 0.6458 - fscore: 0.645 - ETA: 1:03 - loss: 0.6550 - accuracy: 0.6429 - fscore: 0.642 - ETA: 1:02 - loss: 0.6521 - acc
uracy: 0.6328 - fscore: 0.632 - ETA: 1:02 - loss: 0.6560 - accuracy: 0.6181 - fscore: 0.618 - ETA: 1:02 - loss: 0.6456 - accura
cy: 0.6375 - fscore: 0.637 - ETA: 1:02 - loss: 0.6481 - accuracy: 0.6364 - fscore: 0.636 - ETA: 1:01 - loss: 0.6501 - accuracy:
0.6250 - fscore: 0.625 - ETA: 1:00 - loss: 0.6504 - accuracy: 0.6250 - fscore: 0.625 - ETA: 59s - loss: 0.6466 - accuracy: 0.63
84 - fscore: 0.638 - ETA: 58s - loss: 0.6455 - accuracy: 0.6375 - fscore: 0.63 - ETA: 58s - loss: 0.6479 - accuracy: 0.6328 - f
score: 0.63 - ETA: 57s - loss: 0.6481 - accuracy: 0.6287 - fscore: 0.62 - ETA: 56s - loss: 0.6496 - accuracy: 0.6250 - fscore:
```

Fig: 5.15

For the model training, I adopted Reduce Learning On Plateau and saved the best model with the min val_loss only. And here is the model performance of different target class setups.

New Model Performance

-Male 5 Class

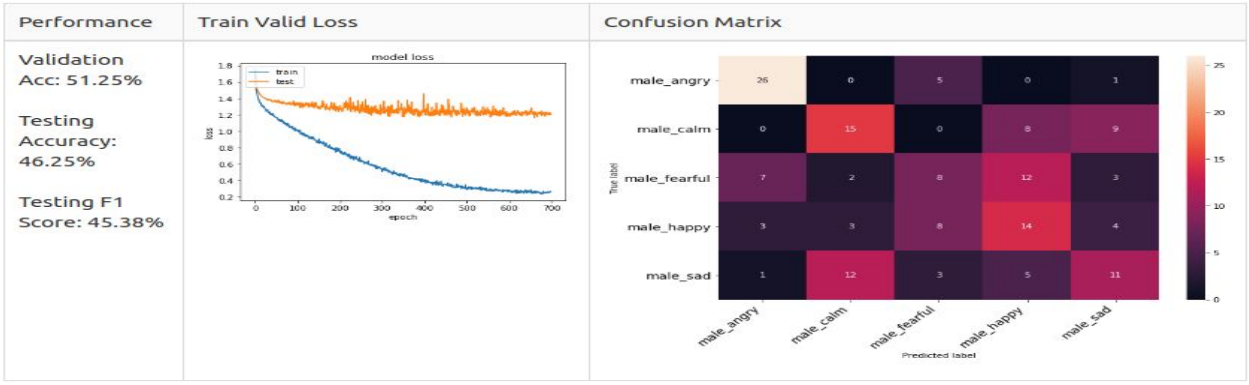


Fig: 5.16

-Female 5 Class

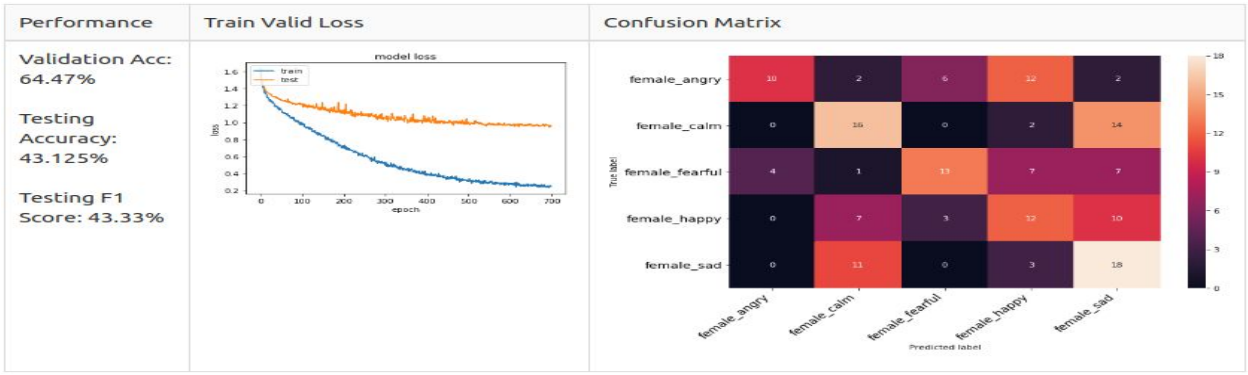


Fig: 5.17

-Male 2 Class

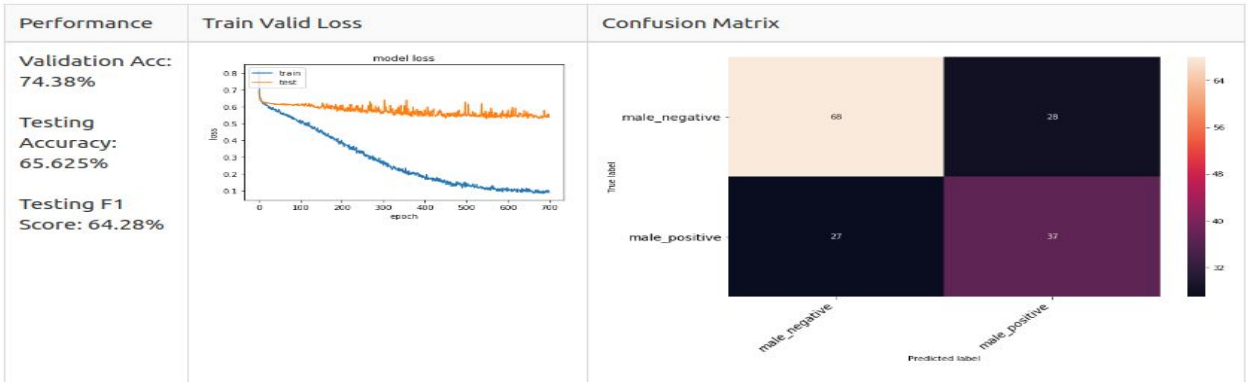
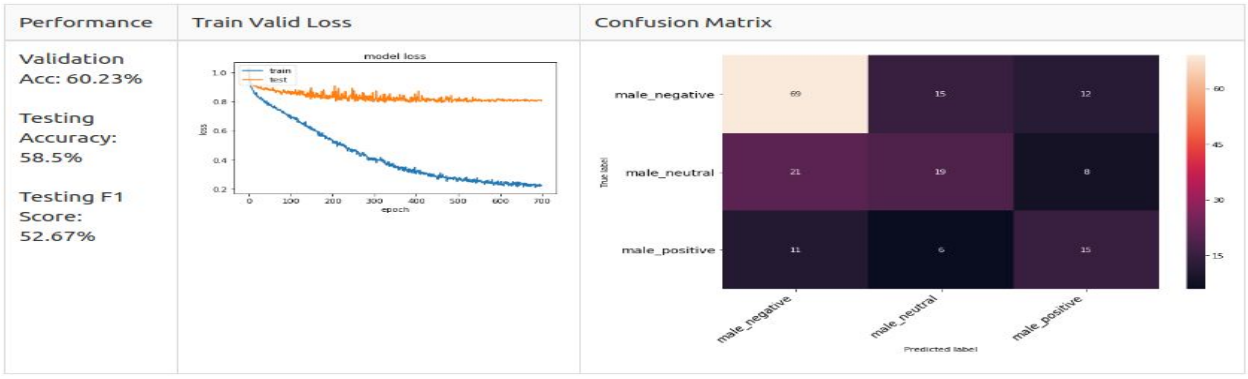


Fig: 5.18

-Male 3 Class



\

Fig: 5.19

Augmentation

After I tuned the model architecture, optimizer and learning rate schedule, I found out the model still cannot converge in the training period. I assumed it is the data size problem since we have 800 samples for train valid set only. Thus, I decided to explore the audio augmentation methods. Let's take a look at some augmentation method with code. I simply augmented all of the datasets once to double the train / valid set size.

Male 5 Class:

Dynamic Value Change

```
def dyn_change(data):
  """
  Random Value Change.
  """
  dyn_change = np.random.uniform(low=1.5,high=3)
  return (data * dyn_change)
```

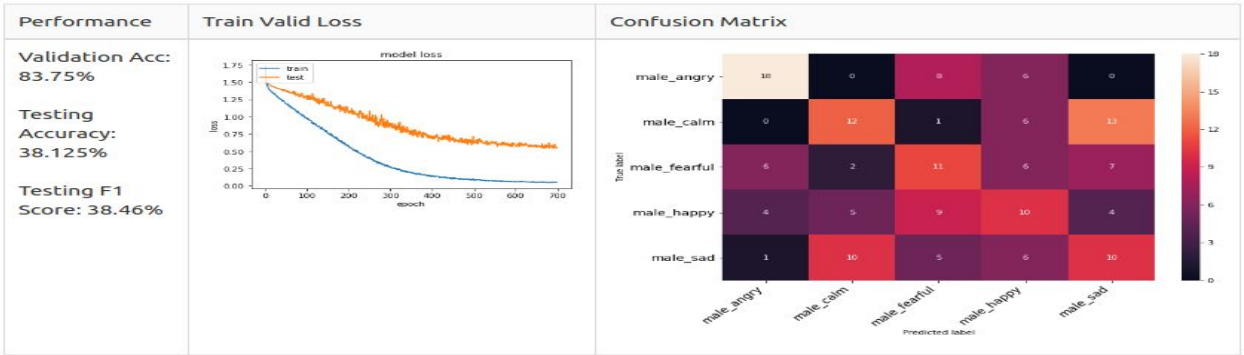


Fig: 5.20

Pitch Tuning

```
def pitch(data, sample_rate):
  """
  Pitch Tuning.
  """
  bins_per_octave = 12
  pitch_pm = 2
  pitch_change =  pitch_pm * 2*(np.random.uniform())
  data = librosa.effects.pitch_shift(data.astype('float64'),
                                     sample_rate, n_steps=pitch_change,
                                     bins_per_octave=bins_per_octave)
```

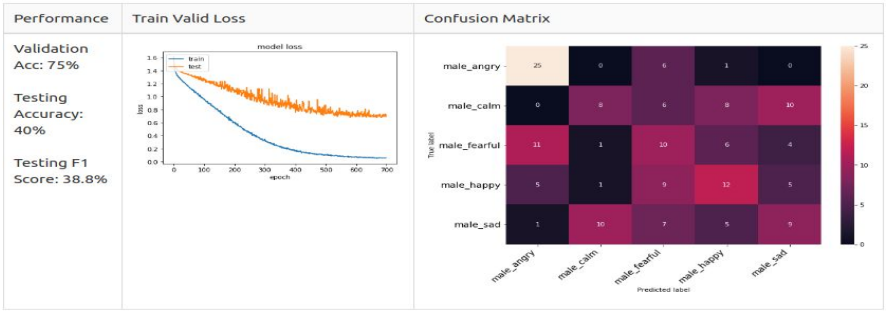


Fig: 5.21

Shifting

```
def shift(data):
  """
  Random Shifting.
  """
  s_range = int(np.random.uniform(low=-5, high = 5)*500)
  return np.roll(data, s_range)
```

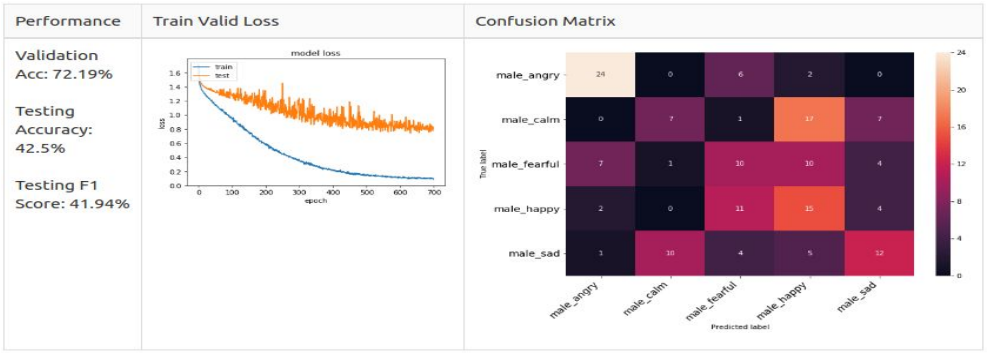


Fig: 5.22

White Noise Adding

```
def noise(data):
    """
    Adding White Noise.
    """
    # you can take any distribution from
https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.random.html
    noise_amp = 0.005*np.random.uniform()*np.amax(data)
    data = data.astype('float64') + noise_amp * np.random.normal(size=data.shape[0])
    return data
```

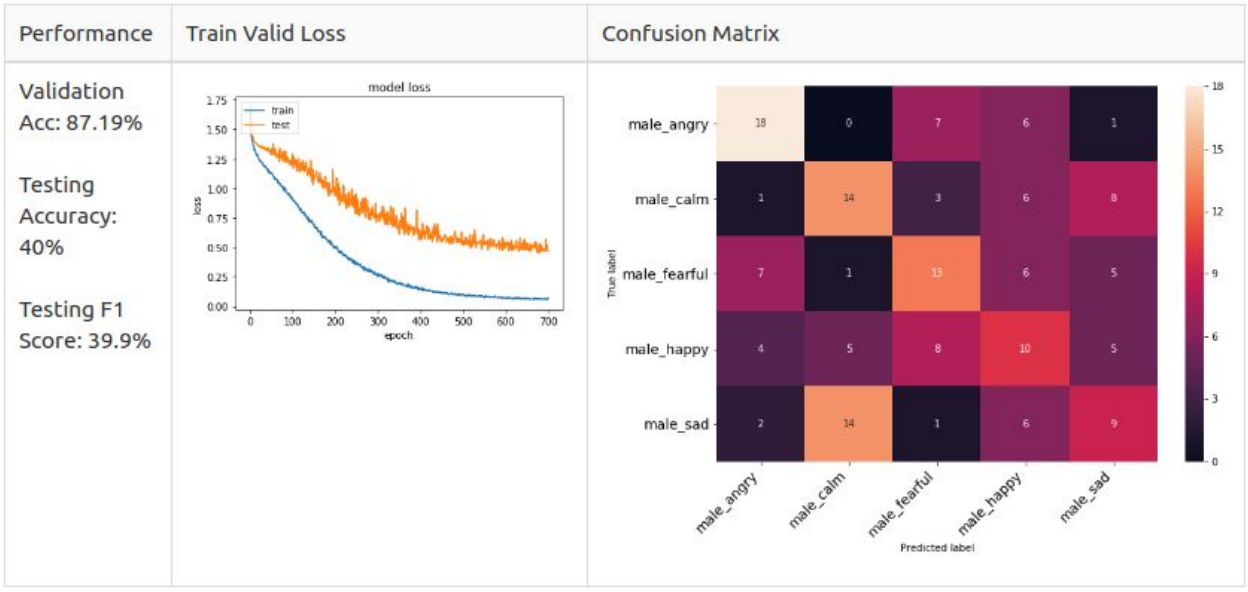


Fig: 5.23

```
In [37]: def plot_time_series(data):
             """
             Plot the Audio Frequency.
             """
             fig = plt.figure(figsize=(14, 8))
             plt.title('Raw wave ')
             plt.ylabel('Amplitude')
             plt.plot(np.linspace(0, 1, len(data)), data)
             plt.show()


         def noise(data):
             """
             Adding White Noise.
             """
             # we can take any distribution from https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.random.html
             noise_amp = 0.005*np.random.uniform()*np.amax(data)
             data = data.astype('float64') + noise_amp * np.random.normal(size=data.shape[0])
             return data

         def shift(data):
             """
             Random Shifting.
             """
             s_range = int(np.random.uniform(low=-5, high = 5)*500)
             return np.roll(data, s_range)

         def stretch(data, rate=0.8):
             """
             Streching the Sound.
             """
             data = librosa.effects.time_stretch(data, rate)
             return data

         def pitch(data, sample_rate):
             """
             Pitch Tuning.
             """
```

```
    Streching the Sound.
    """
    data = librosa.effects.time_stretch(data, rate)
    return data

def pitch(data, sample_rate):
    """
    Pitch Tuning.
    """
    bins_per_octave = 12
    pitch_pm = 2
    pitch_change =  pitch_pm * 2*(np.random.uniform())
    data = librosa.effects.pitch_shift(data.astype('float64'),
                                       sample_rate, n_steps=pitch_change,
                                       bins_per_octave=bins_per_octave)
    return data

def dyn_change(data):
    """
    Random Value Change.
    """
    dyn_change = np.random.uniform(low=1.5,high=3)
    return (data * dyn_change)

def speedNpitch(data):
    """
    peed and Pitch Tuning.
    """
    # we can change low and high here
    length_change = np.random.uniform(low=0.8, high = 1)
    speed_fac = 1.0  / length_change
    tmp = np.interp(np.arange(0,len(data),speed_fac),np.arange(0,len(data)),data)
    minlen = min(data.shape[0], tmp.shape[0])
    data *= 0
    data[0:minlen] = tmp[0:minlen]
    return data
```

Fig: 5.24

We can see that the augmentation can jack up the Validation Accuracy a lot, 70+% in general. Especially that adding white noise can achieve **80.21% Validation Accuracy**, however, the Testing Accuracy and Testing F1-score dropped more than 5% respectively. **i.e:- 75.21 %** ...Then, I wonder if mixing different augmentation methods would bring a good result.

Mixing Multiple Methods

Noise Adding + Shifting

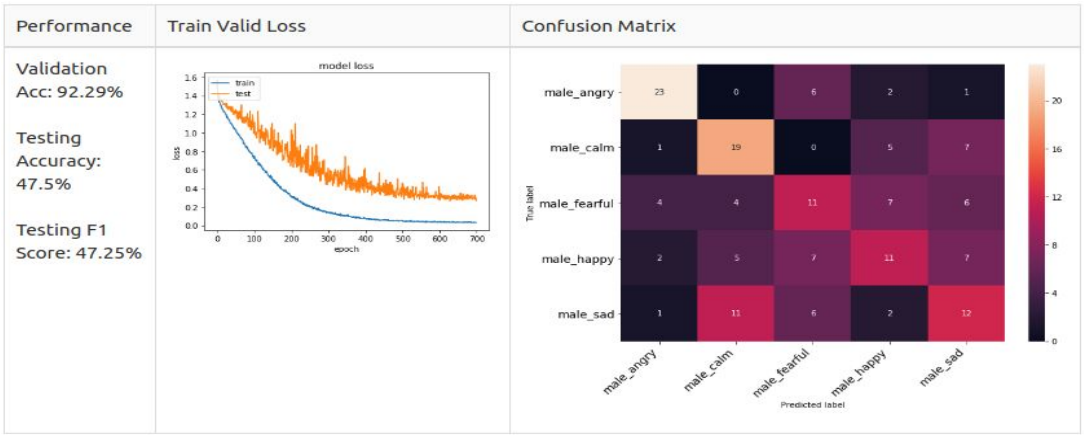


Fig: 5.25

Testing Augmentation on Male 2 Class Data

Male 2 Class:

Noise Adding + Shifting

For all sample

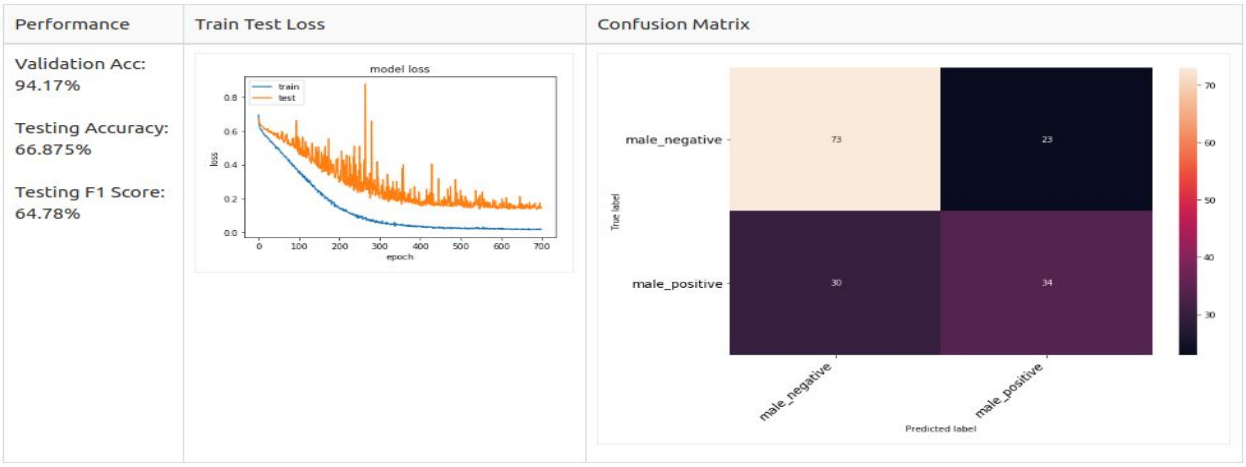


Fig: 5.26

Noise Adding + Shifting

For positive samples only since the 2 class set is imbalance (skewed toward negative).

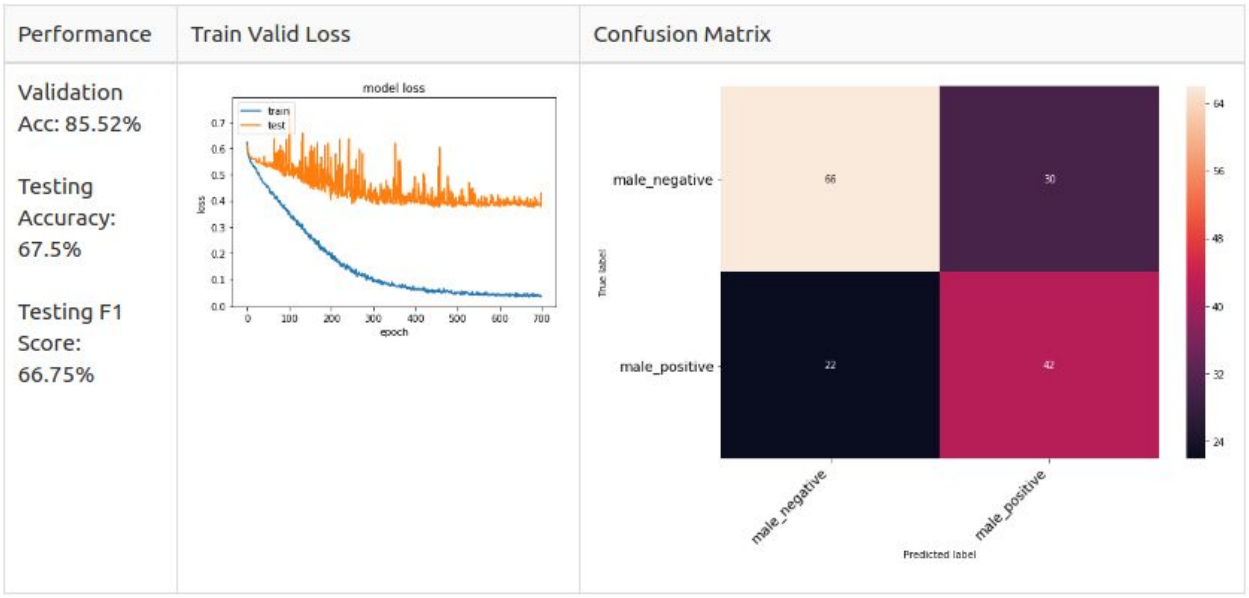


Fig: 5.27

Pitch Tuning + Noise Adding

For all sample

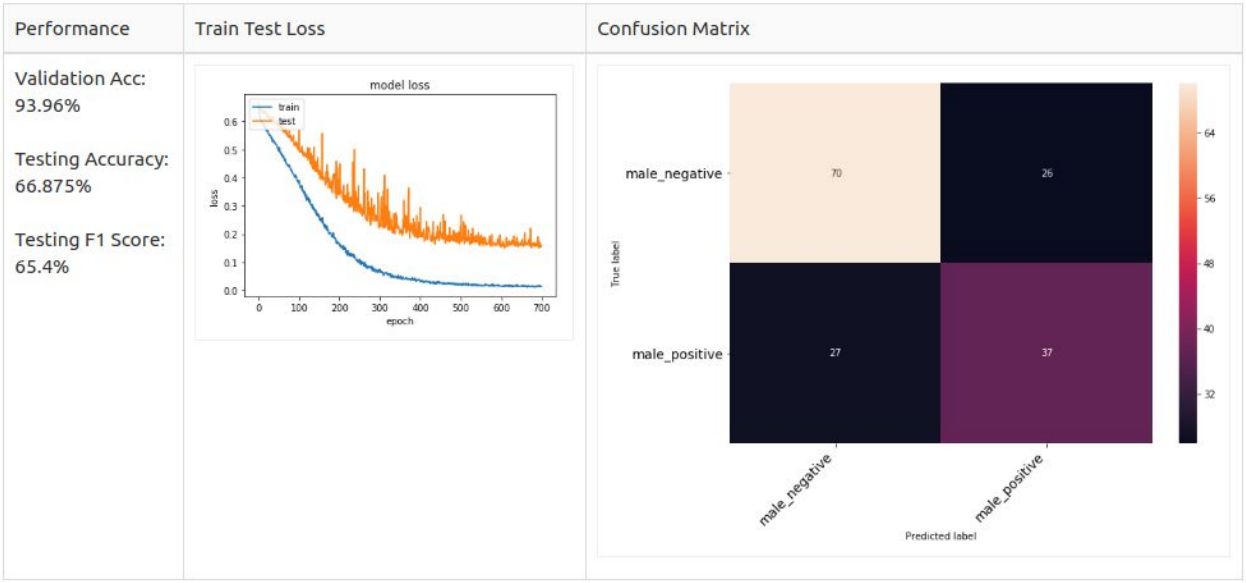


Fig: 5.28

Pitch Tuning + Noise Adding

For positive sample only

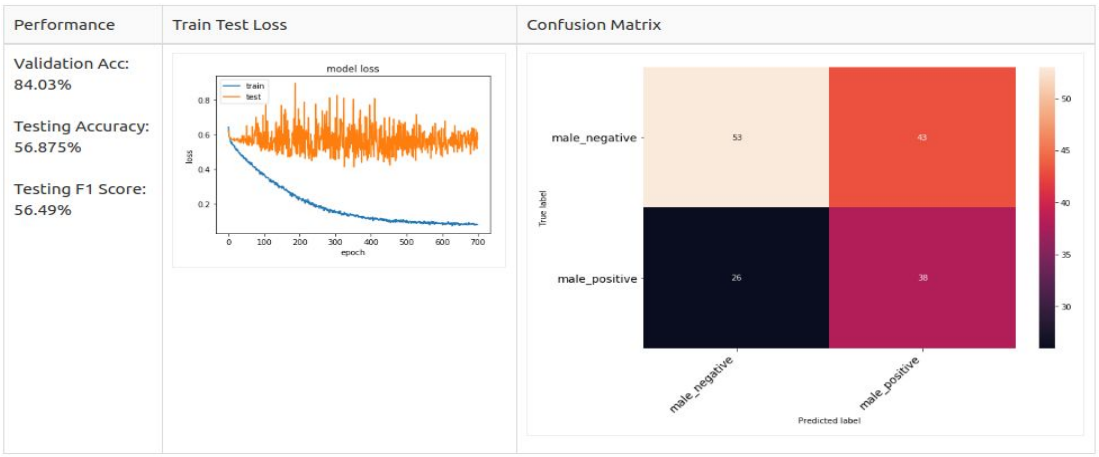


Fig: 5.29

Testing out with live voices

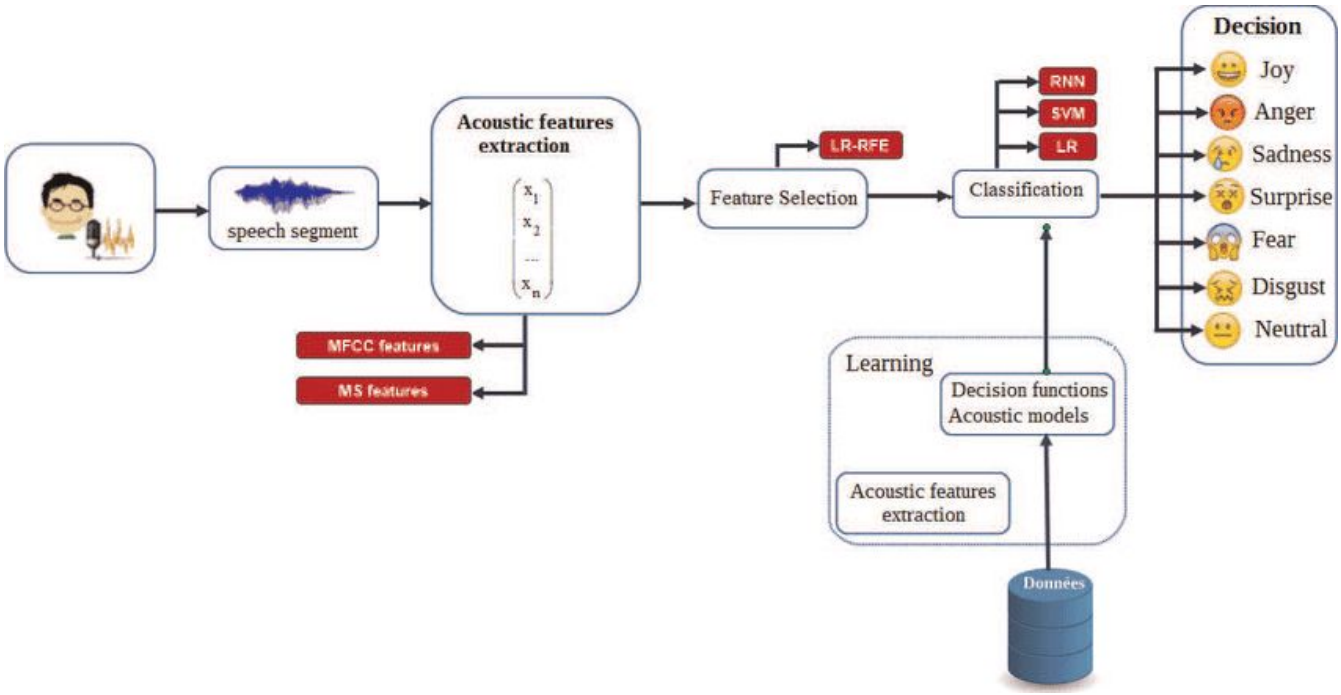


Fig: 5.30

In order to test out our model on voices that were completely different than what we have in our training and test data, we recorded our own voices with different emotions and predicted the outcomes. You can see the results below:

The audio contained a male voice which said **"Your Work Is Genius ..Its Quality Work…." in an angry tone**.

SCREENSHOTS:

Importing the required libraries

In [299]:
```
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from matplotlib.pyplot import specgram
import keras
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
from keras.models import Model
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import confusion_matrix
```

In [300]:
```
from keras import regularizers
```

In [301]:
```
import os
```

In [302]:
```
mylist= os.listdir('C://Users//pranav//Desktop//SER//x2data')
```

In [303]:
```
type(mylist)
```

Out[303]:
```
list
```

In [304]:
```
print(mylist[1800])
```

```
03-02-03-01-02-02-16.wav
```

In [305]:
```
print(mylist[400][6:-16])
```

```
03
```

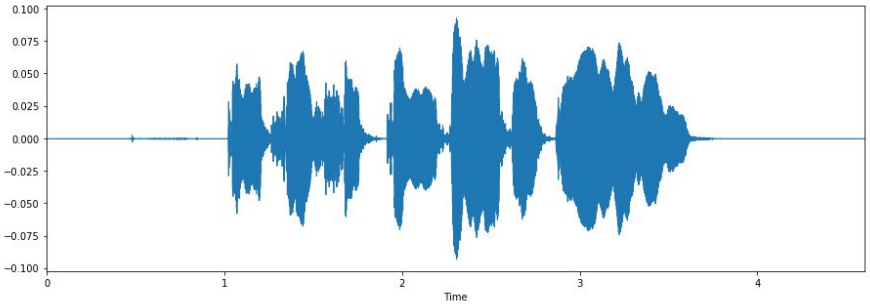
Plotting the audio file's waveform and its spectrogram

In [306]:
```
data, sampling_rate = librosa.load('C://Users//pranav//Desktop//SER//x2data//03-02-03-01-02-02-16.wav')
```

In [307]:
```
%pylab inline
import os
import pandas as pd
import librosa
import glob

plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)
```

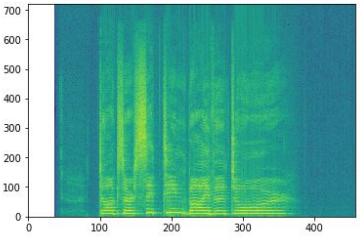
```
Populating the interactive namespace from numpy and matplotlib
F:\New folder\lib\site-packages\IPython\core\magics\pylab.py:160: UserWarning:

pylab import has clobbered these variables: ['shuffle', 'test']
`%matplotlib` prevents importing * from pylab and numpy
```

Out[307]: <matplotlib.collections.PolyCollection at 0x2161caabb38>



In [308]:
```
import matplotlib.pyplot as plt
import scipy.io.wavfile
import numpy as np
import sys


sr,x = scipy.io.wavfile.read('C://Users//pranav//Desktop//SER//x2data//03-02-03-01-02-02-16.wav')

## Parameters: 10ms step, 30ms window
nstep = int(sr * 0.01)
nwin  = int(sr * 0.03)
nfft = nwin

window = np.hamming(nwin)

## will take windows x[n1:n2].  generate
## and loop over n2 such that all frames
## fit within the waveform
nn = range(nwin, len(x), nstep)

X = np.zeros( (len(nn), nfft//2) )

for i,n in enumerate(nn):
    xseg = x[n-nwin:n]
    z = np.fft.fft(window * xseg, nfft)
    X[i,:] = np.log(np.abs(z[:nfft//2]))

plt.imshow(X.T, interpolation='nearest',
    origin='lower',
    aspect='auto')

plt.show()
```

```
F:\New folder\lib\site-packages\scipy\io\wavfile.py:273: WavFileWarning:

Chunk (non-data) not understood, skipping it.

F:\New folder\lib\site-packages\scipy\io\wavfile.py:273: WavFileWarning:

Chunk (non-data) not understood, skipping it.

F:\New folder\lib\site-packages\scipy\io\wavfile.py:273: WavFileWarning:

Chunk (non-data) not understood, skipping it.

F:\New folder\lib\site-packages\ipykernel_launcher.py:26: RuntimeWarning:

divide by zero encountered in log
```



Setting the labels

In [309]:
```
feeling_list=[]
for item in mylist:
    if item[6:-16]=='02' and int(item[18:-4])%2==0:
        feeling_list.append('female_calm')
    elif item[6:-16]=='02' and int(item[18:-4])%2==1:
        feeling_list.append('male_calm')
    elif item[6:-16]=='03' and int(item[18:-4])%2==0:
        feeling_list.append('female_happy')
    elif item[6:-16]=='03' and int(item[18:-4])%2==1:
        feeling_list.append('male_happy')
    elif item[6:-16]=='04' and int(item[18:-4])%2==0:
        feeling_list.append('female_sad')
    elif item[6:-16]=='04' and int(item[18:-4])%2==1:
        feeling_list.append('male_sad')
    elif item[6:-16]=='05' and int(item[18:-4])%2==0:
        feeling_list.append('female_angry')
    elif item[6:-16]=='05' and int(item[18:-4])%2==1:
        feeling_list.append('male_angry')
    elif item[6:-16]=='06' and int(item[18:-4])%2==0:
        feeling_list.append('female_fearful')
    elif item[6:-16]=='06' and int(item[18:-4])%2==1:
        feeling_list.append('male_fearful')
    elif item[:1]=='a':
        feeling_list.append('male_angry')
    elif item[:1]=='f':
        feeling_list.append('male_fearful')
    elif item[:1]=='h':
        feeling_list.append('male_happy')
    #elif item[:1]=='n':
        #feeling_list.append('neutral')
    elif item[:2]=='sa':
        feeling_list.append('male_sad')
```

In [310]:
```
labels = pd.DataFrame(feeling_list)
```

In [311]:
```
labels[:10]
```

Out[311]:

	0
0	male_calm
1	female_calm
2	male_calm
3	female_calm
4	male_calm
5	female_calm
6	male_calm
7	female_calm
8	male_calm
9	female_calm

Getting the features of audio files using librosa

In [312]:
```
df = pd.DataFrame(columns=['feature'])
bookmark=0
for index,y in enumerate(mylist):
    if mylist[index][6:-16]!='01' and mylist[index][6:-16]!='07' and mylist[index][6:-16]!='08' and mylist[index][:2]!='su' and 
        X, sample_rate = librosa.load('C://Users//pranav//Desktop//SER//x2data//'+y, res_type='kaiser_fast',duration=2.5,sr=2205
        sample_rate = np.array(sample_rate)
        mfccs = np.mean(librosa.feature.mfcc(y=X,
                                             sr=sample_rate,
                                             n_mfcc=13),
                        axis=0)
        feature = mfccs
        #[float(i) for i in feature]
        #feature1=feature[:135]
        df.loc[bookmark] = [feature]
        bookmark=bookmark+1
```

In [313]:
```
df[:5]
```

Out[313]:

	feature
0	[-70.2677641610773, -70.2677641610773, -70.267...
1	[-65.70765240065282, -65.70765240065282, -63.1...
2	[-65.4824988827423, -65.4824988827423, -65.482...
3	[-64.52844910346735, -64.52844910346735, -64.5...
4	[-62.36431052745468, -59.93472513811134, -61.8...

In [314]:
```
df3 = pd.DataFrame(df['feature'].values.tolist())
```

df3[:5]

```
In [315]: newdf = pd.concat([df3,labels], axis=1)
```

```
In [316]: rnewdf = newdf.rename(index=str, columns={"0": "label"})
```

```
In [317]: rnewdf[:5]
```

Out[317]:

	0	1	2	3	4	5	6	7	8	9	...	207	208	209
0	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	...	-57.447461	-58.896493	-58.751002
1	-65.707652	-65.707652	-63.114722	-61.518999	-61.097138	-63.424602	-63.720067	-56.854608	-55.168972	-54.640002	...	-39.792147	-40.613166	-41.209201
2	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	-65.482499	...	-31.346553	-34.310774	-35.800705
3	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-64.528449	-65.928222	...	-48.674306	-48.596082	-47.602751
4	-62.364311	-59.934725	-61.869600	-67.495764	-71.071811	-65.679826	-63.394396	-65.503349	-61.856639	-60.005421	...	-39.071328	-41.897121	-40.865430

5 rows × 217 columns

```
In [318]: from sklearn.utils import shuffle
          rnewdf = shuffle(newdf)
          rnewdf[:10]
```

Out[318]:

	0	1	2	3	4	5	6	7	8	9	...	207	208	209
379	-42.631395	-42.631395	-42.631395	-42.631395	-42.631395	-42.631395	-42.631395	-42.631395	-42.631395	-42.631395	...	-42.631395	-42.631395	-42.6313
876	-64.311404	-58.837348	-57.070674	-59.403286	-59.159440	-56.119573	-56.304894	-62.215263	-62.511671	-64.500209	...	-60.223600	-57.189248	-57.9628
451	-79.090368	-79.090368	-79.090368	-79.090368	-79.090368	-79.090368	-79.090368	-79.090368	-79.090368	-79.090368	...	-63.005665	-67.966180	-66.2692
1480	-49.250083	-47.862935	-47.617017	-49.317434	-47.837021	-45.822780	-43.965543	-43.504021	-44.797301	-46.603719	...	-29.019142	-29.287913	-30.0685
547	-65.465481	-65.465481	-65.465481	-65.465481	-65.465481	-65.465481	-65.465481	-65.465481	-65.465481	-65.465481	...	-65.465922	-64.065300	-64.0868
390	-71.188637	-71.188637	-71.188637	-71.188637	-71.188637	-69.602972	-68.462377	-67.832988	-65.815999	-65.290115	...	-59.432881	-57.374782	-55.1246
911	-46.869321	-46.869321	-46.869321	-46.869321	-46.869321	-46.869321	-46.869321	-46.869321	-46.869321	-46.869321	...	-37.961509	-39.621291	-40.2184
597	-53.678315	-54.377914	-55.140844	-56.300240	-55.326932	-55.119810	-56.282534	-54.970338	-53.334915	-52.519249	...	-50.354847	-51.740110	-53.1516
1326	-49.875624	-49.875624	-49.875624	-48.006241	-47.245683	-47.207803	-47.366410	-47.511315	-47.409282	-47.263355	...	-26.248085	-26.273129	-26.8256
974	-51.100772	-49.108443	-49.629691	-52.753706	-54.096657	-55.859434	-54.466789	-55.492749	-57.559655	-57.487213	...	-31.491357	-31.564764	-30.8826

10 rows × 217 columns

```
In [319]: rnewdf=rnewdf.fillna(0)
```

Dividing the data into test and train

```
In [321]: newdf1 = np.random.rand(len(rnewdf)) < 0.8
          train = rnewdf[newdf1]
          test = rnewdf[~newdf1]
```

```
In [322]: train[250:260]
```

Out[322]:

	0	1	2	3	4	5	6	7	8	9	...	207	208	209
1141	-53.940489	-54.585260	-52.693097	-52.662991	-53.539774	-53.312791	-53.658248	-53.033167	-52.704671	-51.875424	...	-44.746416	-42.721836	-42.0
779	-53.536973	-54.117284	-54.527581	-54.527581	-52.853619	-52.647446	-52.572823	-52.529147	-53.306279	-53.073154	...	-45.253403	-44.261883	-44.0
664	-54.071434	-53.270441	-55.962005	-55.425948	-56.699237	-55.350538	-52.082626	-53.349495	-54.608616	-54.911032	...	-53.482203	-53.487210	-52.9
827	-57.419633	-57.696347	-58.593154	-58.173707	-56.816082	-57.052646	-56.534252	-55.248651	-54.070975	-53.961269	...	-50.393697	-50.474974	-51.0
1686	-49.004734	-49.004734	-49.004734	-49.004734	-49.004734	-49.004734	-49.004734	-49.004734	-49.004734	-49.004734	...	-32.870681	-35.278584	-36.0
312	-56.733879	-56.752659	-56.415251	-55.715674	-55.817854	-55.832755	-55.889748	-55.116522	-54.244677	-56.211891	...	-39.931887	-43.064495	-46.6
1666	-42.853688	-43.125471	-43.261603	-43.542698	-43.996154	-44.278831	-43.174748	-43.805186	-44.280954	-44.266471	...	-17.316529	-14.849276	-12.0
1565	-51.814071	-51.179072	-51.548308	-50.435058	-49.806378	-50.147107	-49.875816	-50.006132	-51.929632	-50.753609	...	-32.295638	-32.723007	-34.4
542	-60.879858	-60.879858	-60.879858	-59.529157	-57.068292	-56.470826	-58.132601	-60.712885	-60.090474	-58.996574	...	-47.326807	-46.939704	-48.4
477	-52.832334	-52.688440	-52.211688	-51.576756	-51.757801	-51.686870	-52.709010	-53.476120	-54.586895	-56.060488	...	-51.755319	-52.064901	-49.5

```
In [323]: trainfeatures = train.iloc[:, :-1]
```

```
In [324]: trainlabel = train.iloc[:, -1:]
```

In [325]:
```
testfeatures = test.iloc[:, :-1]
```

In [326]:
```
testlabel = test.iloc[:, -1:]
```

In [327]:
```
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder

X_train = np.array(trainfeatures)
y_train = np.array(trainlabel)
X_test = np.array(testfeatures)
y_test = np.array(testlabel)

lb = LabelEncoder()

y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))
```

```
F:\New folder\lib\site-packages\sklearn\preprocessing\_label.py:251: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using rav
el().

F:\New folder\lib\site-packages\sklearn\preprocessing\_label.py:251: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using rav
el().
```

In [328]:
```
y_train
```

Out[328]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

In [329]:
```
y_test
```

Out[329]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 1., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

In [330]:
```
X_train.shape
```

Out[330]:
```
(1525, 216)
```

In [331]:
```
shape(X_train)
```

Out[331]:
```
(1525, 216)
```

In [332]:
```
 shape(y_train)
```

Out[332]:
```
(1525, 10)
```

In [333]:
```
shape(X_test)
```

Out[333]:
```
(355, 216)
```

In [334]:
```
shape(y_test)
```

Out[334]:
```
(355, 10)
```

In [335]:
```
rnewdf=rnewdf.fillna(0)
```

Changing dimension for CNN model

In [336]:
```
x_traincnn =np.expand_dims(X_train, axis=2)
x_testcnn= np.expand_dims(X_test, axis=2)
```

In [337]:
```
model = Sequential()

model.add(Conv1D(256, 5,padding='same',
                 input_shape=(216,1)))
model.add(Activation('relu'))
model.add(Conv1D(128, 5,padding='same'))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(128, 5,padding='same',))
model.add(Activation('relu'))
#model.add(Conv1D(128, 5,padding='same',))
#model.add(Activation('relu'))
#model.add(Conv1D(128, 5,padding='same',))
#model.add(Activation('relu'))
#model.add(Dropout(0.2))
model.add(Conv1D(128, 5,padding='same',))
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dense(10))
model.add(Activation('softmax'))
opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
```

In [338]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_13 (Conv1D)           (None, 216, 256)          1536
_________________________________________________________________
activation_16 (Activation)   (None, 216, 256)          0
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 216, 128)          163968
_________________________________________________________________
activation_17 (Activation)   (None, 216, 128)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 216, 128)          0
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 27, 128)           0
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_18 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_19 (Activation)   (None, 27, 128)           0
_________________________________________________________________
flatten_4 (Flatten)          (None, 3456)              0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                34570
_________________________________________________________________
activation_20 (Activation)   (None, 10)                0
=================================================================
Total params: 364,170
Trainable params: 364,170
Non-trainable params: 0
_________________________________________________________________
```

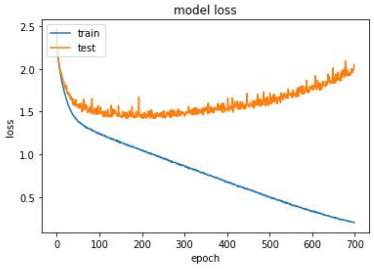
In [339]:
```
model.compile(loss='categorical_crossentropy', optimizer=opt,metrics=['accuracy'])
```

Removed the whole training part for avoiding unnecessary long epochs list

In [340]:
```
cnnhistory=model.fit(x_traincnn, y_train, batch_size=16, epochs=700, validation_data=(x_testcnn, y_test))
```

```
WARNING:tensorflow:From F:\New folder\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.pyt
hon.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 1525 samples, validate on 355 samples
Epoch 1/700
1525/1525 [==============================] - ETA: 3:07 - loss: 4.5898 - accuracy: 0.06 - ETA: 1:40 - loss: 3.9476 - accuracy:
0.09 - ETA: 1:10 - loss: 3.7577 - accuracy: 0.08 - ETA: 56s - loss: 3.7785 - accuracy: 0.0781 - ETA: 47s - loss: 3.6515 - acc
uracy: 0.087 - ETA: 41s - loss: 3.5898 - accuracy: 0.093 - ETA: 36s - loss: 3.4819 - accuracy: 0.098 - ETA: 33s - loss: 3.409
6 - accuracy: 0.109 - ETA: 31s - loss: 3.3362 - accuracy: 0.111 - ETA: 29s - loss: 3.2897 - accuracy: 0.112 - ETA: 27s - los
s: 3.2495 - accuracy: 0.113 - ETA: 25s - loss: 3.1958 - accuracy: 0.114 - ETA: 24s - loss: 3.1728 - accuracy: 0.115 - ETA: 23
s - loss: 3.1437 - accuracy: 0.111 - ETA: 22s - loss: 3.1072 - accuracy: 0.104 - ETA: 21s - loss: 3.0825 - accuracy: 0.109 -
ETA: 20s - loss: 3.0476 - accuracy: 0.106 - ETA: 19s - loss: 3.0248 - accuracy: 0.104 - ETA: 19s - loss: 3.0049 - accuracy:
0.105 - ETA: 18s - loss: 2.9796 - accuracy: 0.112 - ETA: 17s - loss: 2.9661 - accuracy: 0.107 - ETA: 17s - loss: 2.9471 - acc
uracy: 0.105 - ETA: 16s - loss: 2.9159 - accuracy: 0.103 - ETA: 16s - loss: 2.8997 - accuracy: 0.101 - ETA: 15s - loss: 2.885
1 - accuracy: 0.100 - ETA: 15s - loss: 2.8669 - accuracy: 0.096 - ETA: 15s - loss: 2.8492 - accuracy: 0.094 - ETA: 14s - los
s: 2.8397 - accuracy: 0.093 - ETA: 14s - loss: 2.8258 - accuracy: 0.097 - ETA: 14s - loss: 2.8110 - accuracy: 0.100 - ETA: 13
s - loss: 2.7971 - accuracy: 0.096 - ETA: 13s - loss: 2.7817 - accuracy: 0.095 - ETA: 13s - loss: 2.7727 - accuracy: 0.094 -
ETA: 12s - loss: 2.7640 - accuracy: 0.091 - ETA: 12s - loss: 2.7567 - accuracy: 0.089 - ETA: 12s - loss: 2.7412 - accuracy:
```

In [341]:
```
plt.plot(cnnhistory.history['loss'])
plt.plot(cnnhistory.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Saving the model

In [342]:
```
model_name = 'Emotion_Voice_Detection_Model.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')
# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
```

```
Saved trained model at C:\Users\pranav\Desktop\saved_models\Emotion_Voice_Detection_Model.h5
```

In [343]:
```
import json
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

Loading the model

In [345]:
```
# loading json and creating model
from keras.models import model_from_json
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("saved_models/Emotion_Voice_Detection_Model.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
score = loaded_model.evaluate(x_testcnn, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
Loaded model from disk
accuracy: 44.79%
```

Predicting emotions on the test data

In [347]:
```
preds = loaded_model.predict(x_testcnn,
                             batch_size=32,
                             verbose=1)
```

```
355/355 [==============================] - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:
- 1s 3ms/step
```

In [348]:
```
preds
```

Out[348]:
```
array([[1.4473349e-07, 9.9864608e-01, 1.7444214e-07, ..., 2.9179098e-10,
        4.5685247e-09, 3.3200675e-07],
       [3.3178358e-04, 2.5577401e-03, 1.0866145e-05, ..., 3.0719312e-02,
        2.6371932e-01, 6.1018842e-01],
       [3.9605628e-05, 1.8363297e-08, 8.4159339e-08, ..., 6.4284939e-01,
        1.6855605e-01, 1.3364529e-02],
       ...,
       [9.4780858e-05, 2.0215945e-17, 1.5030613e-09, ..., 7.4200170e-06,
        7.7437602e-05, 1.6252535e-08],
       [1.0328042e-04, 5.2118665e-03, 1.3182412e-07, ..., 8.6186573e-02,
        1.7135671e-01, 5.6733614e-01],
       [9.9996126e-01, 1.7521634e-08, 3.2920103e-05, ..., 4.5457518e-09,
        4.5314405e-06, 9.2682179e-10]], dtype=float32)
```

In [349]:
```
preds1=preds.argmax(axis=1)
```

In [350]:
```
preds1
```

Out[350]:
```
array([1, 9, 7, 6, 6, 9, 3, 9, 0, 7, 0, 1, 0, 1, 9, 3, 1, 8, 2, 1, 0, 8,
       3, 0, 5, 2, 5, 5, 0, 1, 4, 6, 6, 5, 4, 9, 3, 1, 1, 9, 5, 9, 1, 9,
       0, 9, 9, 6, 1, 3, 9, 9, 1, 1, 3, 1, 9, 4, 7, 2, 5, 6, 0, 4, 1, 6,
       1, 8, 9, 6, 2, 8, 3, 0, 7, 9, 7, 1, 7, 4, 0, 3, 1, 1, 0, 8, 6, 5,
       3, 9, 3, 2, 3, 4, 6, 1, 6, 7, 7, 1, 5, 5, 9, 0, 8, 6, 6, 9, 6, 3,
       3, 9, 8, 2, 9, 3, 0, 5, 7, 1, 7, 0, 1, 2, 1, 4, 0, 6, 3, 9, 2, 1,
       4, 7, 2, 8, 8, 1, 1, 4, 4, 4, 3, 3, 4, 2, 1, 5, 9, 2, 6, 7, 0, 7,
       3, 0, 0, 1, 1, 1, 1, 0, 2, 6, 8, 1, 0, 5, 4, 8, 5, 6, 0, 6, 4, 7,
       8, 9, 0, 8, 2, 1, 2, 7, 9, 1, 6, 4, 9, 7, 3, 5, 7, 3, 9, 5, 2, 5,
       1, 7, 6, 1, 8, 9, 0, 1, 1, 1, 4, 7, 6, 7, 8, 1, 3, 2, 8, 5, 5, 3,
       7, 1, 1, 1, 8, 7, 2, 4, 5, 2, 5, 2, 8, 5, 5, 7, 2, 8, 4, 1, 1, 2,
       5, 7, 7, 0, 0, 8, 1, 3, 5, 0, 5, 7, 5, 5, 0, 5, 9, 1, 9, 5, 0, 5,
       8, 2, 0, 9, 2, 3, 9, 1, 0, 6, 1, 6, 6, 1, 1, 6, 4, 2, 1, 1, 6, 4,
       4, 0, 7, 2, 7, 1, 1, 8, 9, 7, 9, 1, 7, 1, 9, 8, 0, 9, 9, 8, 8, 7,
       9, 7, 9, 0, 3, 0, 9, 3, 1, 9, 2, 5, 3, 7, 0, 1, 1, 7, 0, 1, 3, 1,
       8, 9, 7, 5, 5, 2, 8, 3, 7, 5, 5, 1, 3, 4, 9, 9, 7, 1, 9, 9, 8, 5,
       5, 9, 0], dtype=int64)
```

In [351]:
```
abc = preds1.astype(int).flatten()
```

In [352]:
```
predictions = (lb.inverse_transform((abc)))
```

In [353]:
```
preddf = pd.DataFrame({'predictedvalues': predictions})
preddf[:10]
```

Out[353]:

	predictedvalues
0	female_calm
1	male_sad
2	male_fearful
3	male_calm
4	male_calm
5	male_sad
6	female_happy
7	male_sad
8	female_angry
9	male_fearful

In [354]:
```
actual=y_test.argmax(axis=1)
abc123 = actual.astype(int).flatten()
actualvalues = (lb.inverse_transform((abc123)))
```

In [355]:
```
actualdf = pd.DataFrame({'actualvalues': actualvalues})
actualdf[:10]
```

Out[355]:

	actualvalues
0	female_sad
1	male_sad
2	male_happy
3	male_calm
4	male_happy
5	male_sad

9 male_nappy

In [356]:
```
finaldf = actualdf.join(preddf)
```

Actual v/s Predicted emotions

In [357]:
```
finaldf[170:180]
```

Out[357]:

	actualvalues	predictedvalues
170	male_angry	male_angry
171	male_sad	male_calm
172	female_angry	female_angry
173	male_calm	male_calm
174	female_calm	female_sad
175	male_happy	male_fearful
176	male_happy	male_happy
177	male_sad	male_sad
178	female_angry	female_angry
179	male_happy	male_happy

In [358]:
```
finaldf.groupby('actualvalues').count()
```

Out[358]:

actualvalues	predictedvalues
female_angry	34
female_calm	36
female_fearful	36
female_happy	32
female_sad	41
male_angry	40
male_calm	32
male_fearful	29
male_happy	45
male_sad	30

In [359]:
```
finaldf.groupby('predictedvalues').count()
```

Out[359]:

predictedvalues	actualvalues
female_angry	37
female_calm	64
female_fearful	27
female_happy	30
female_sad	22
male_angry	37
male_calm	27
male_fearful	37
male_happy	28
male_sad	46

In [360]:
```
finaldf.to_csv('Predictions.csv', index=False)
```

Live Demo

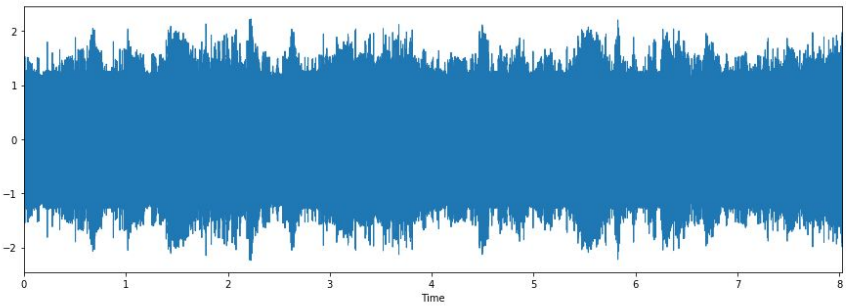
The file 'pranav.wav' in the next cell is the file that was recorded live using the code in AudioRecoreder

In [374]:
```
data, sampling_rate = librosa.load('pranav.wav')
```

In [375]:
```
%pylab inline
import os
import pandas as pd
import librosa
import glob

plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)
```

```
Populating the interactive namespace from numpy and matplotlib
```

Out[375]:
```
<matplotlib.collections.PolyCollection at 0x21621ad5940>
```



In [376]:
```
#livedf= pd.DataFrame(columns=['feature'])
X, sample_rate = librosa.load('pranav.wav', res_type='kaiser_fast',duration=2.5,sr=22050*2,offset=0.5)
sample_rate = np.array(sample_rate)
mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13),axis=0)
featurelive = mfccs
livedf2 = featurelive
```

In [377]:
```
livedf2= pd.DataFrame(data=livedf2)
```

In [378]:
```
livedf2 = livedf2.stack().to_frame().T
```

In [379]:
```
livedf2
```

Out[379]:

	0	1	2	3	4	5	6	7	8	9	...	206	207	208	209	2
	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
0	18.919545	16.221954	15.10257	14.48089	14.975601	13.118913	13.188997	12.068094	10.24041	7.993032	...	10.275322	9.491875	10.813893	15.939725	1

1 rows × 216 columns

In [380]:
```
twodim= np.expand_dims(livedf2, axis=2)
```

In [381]:
```
livepreds = loaded_model.predict(twodim,
                         batch_size=32,
                         verbose=1)
```

```
1/1 [==============================] - 0s 8ms/step
```

In [382]:
```
livepreds
```

Out[382]:
```
array([[2.2344072e-34, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00, 0.0000000e+00, 3.1332897e-35,
        0.0000000e+00, 0.0000000e+00]], dtype=float32)
```

In [383]:
```
livepreds1=livepreds.argmax(axis=1)
```

In [384]:
```
liveabc = livepreds1.astype(int).flatten()
```

In [385]:
```
livepredictions = (lb.inverse_transform((liveabc)))
livepredictions
```

Out[385]:
```
array(['male_angry'], dtype=object)
```

In [ ]:

Fig: 5.31

Chapter 6. Results and Screenshots

In this Python project, we learned to recognize emotions from speech. We used an MLPClassifier for this and made use of the sound file library to read the sound file, and the librosa library to extract features from it.

As you'll see, the model delivered an **accuracy of 75.21%.** That's good enough for us yet.

Accuracy Score: 75.21%

Model has an accuracy of 75.21%, which is considered a good one.

SCREENSHOTS:-

I. Importing the required libraries

In [1]:

```
## Python
import os
import random
import sys


## Package
import glob
import keras
import IPython.display as ipd
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.graph_objs as go
import plotly.offline as py
import plotly.tools as tls
import seaborn as sns
import scipy.io.wavfile
import tensorflow as tf
py.init_notebook_mode(connected=True)


## Keras
from keras import regularizers
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping
from keras.callbacks import  History, ReduceLROnPlateau, CSVLogger
from keras.models import Model, Sequential
from keras.layers import Dense, Embedding, LSTM
from keras.layers import Input, Flatten, Dropout, Activation, BatchNormalization
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
from keras.preprocessing import sequence
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.utils import np_utils
from keras.utils import to_categorical
```

```
## Sklearn
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedShuffleSplit


## Rest
from scipy.fftpack import fft
from scipy import signal
from scipy.io import wavfile
from tqdm import tqdm

input_duration=3
# % pylab inline
```

```
Using TensorFlow backend.
```

II. Reading the data

In [2]:
```
# Data Directory

dir_list = os.listdir('C://Users//pranav//Desktop//SER//xdata//')
dir_list.sort()
print (dir_list)
```

```
['Actor_01', 'Actor_02', 'Actor_03', 'Actor_04', 'Actor_05', 'Actor_06', 'Actor_07', 'Actor_08', 'Actor_09', 'Actor_10', 'Actor
_11', 'Actor_12', 'Actor_13', 'Actor_14', 'Actor_15', 'Actor_16', 'Actor_17', 'Actor_18', 'Actor_19', 'Actor_20', 'Actor_21',
'Actor_22', 'Actor_23', 'Actor_24']
```

In [3]:
```
# Create DataFrame for Data intel

data_df = pd.DataFrame(columns=['path', 'source', 'actor', 'gender',
                                'intensity', 'statement', 'repetition', 'emotion'])
count = 0
for i in dir_list:
    file_list = os.listdir('C://Users//pranav//Desktop//SER//xdata//' + i)
    for f in file_list:
        nm = f.split('.')[0].split('-')
        path = 'C://Users//pranav//Desktop//SER//xdata//' + i + '//' + f
        src = int(nm[1])
        actor = int(nm[-1])
        emotion = int(nm[2])

        if int(actor)%2 == 0:
            gender = "female"
        else:
            gender = "male"

        if nm[3] == '01':
            intensity = 0
        else:
            intensity = 1

        if nm[4] == '01':
            statement = 0
        else:
            statement = 1

        if nm[5] == '01':
            repeat = 0
        else:
            repeat = 1

        data_df.loc[count] = [path, src, actor, gender, intensity, statement, repeat, emotion]
        count += 1
```

In [4]:
```
print (len(data_df))
data_df.head()
```

```
2452
```

Out[4]:

	path	source	actor	gender	intensity	statement	repetition	emotion
0	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	0	0	1
1	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	0	1	1
2	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	1	0	1
3	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	1	1	1
4	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	0	0	2

III. Plotting the audio file's waveform and its spectrogram

In [5]:
```
filename = data_df.path[1021]
print (filename)

samples, sample_rate = librosa.load(filename)
sample_rate, samples
```

```
C://Users//pranav//Desktop//SER//xdata//Actor_10//03-02-04-02-01-02-10.wav
```

Out[5]:
```
(22050, array([-3.4990700e-04, -2.1845367e-04,  5.5546162e-04, ...,
        -9.7345328e-06, -3.8411690e-05,  0.0000000e+00], dtype=float32))
```

In [6]:
```
len(samples), sample_rate
```

Out[6]:
```
(112568, 22050)
```

In [7]:
```
def log_specgram(audio, sample_rate, window_size=20,
                 step_size=10, eps=1e-10):
    nperseg = int(round(window_size * sample_rate / 1e3))
    noverlap = int(round(step_size * sample_rate / 1e3))
    freqs, times, spec = signal.spectrogram(audio,
                                    fs=sample_rate,
                                    window='hann',
                                    nperseg=nperseg,
                                    noverlap=noverlap,
                                    detrend=False)
    return freqs, times, np.log(spec.T.astype(np.float32) + eps)
```

In [8]:
```
sample_rate/ len(samples)
```

Out[8]:
```
0.19588160045483619
```

In [9]:
```
# Plotting Wave Form and Spectrogram

freqs, times, spectrogram = log_specgram(samples, sample_rate)

fig = plt.figure(figsize=(14, 8))
ax1 = fig.add_subplot(211)
ax1.set_title('Raw wave of ' + filename)
ax1.set_ylabel('Amplitude')
librosa.display.waveplot(samples, sr=sample_rate)

ax2 = fig.add_subplot(212)
ax2.imshow(spectrogram.T, aspect='auto', origin='lower',
           extent=[times.min(), times.max(), freqs.min(), freqs.max()])
ax2.set_yticks(freqs[::16])
ax2.set_xticks(times[::16])
ax2.set_title('Spectrogram of ' + filename)
ax2.set_ylabel('Freqs in Hz')
ax2.set_xlabel('Seconds')
```

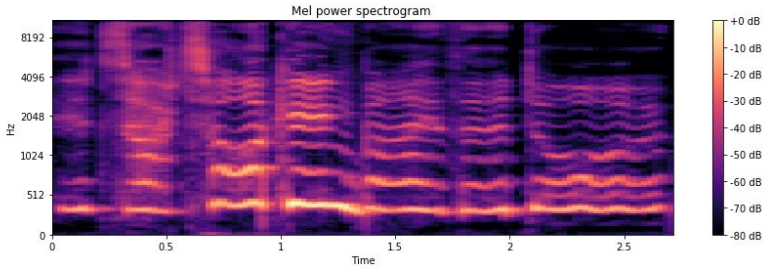
Out[9]:
```
Text(0.5,0,'Seconds')
```

```
In [10]: mean = np.mean(spectrogram, axis=0)
         std = np.std(spectrogram, axis=0)
         spectrogram = (spectrogram - mean) / std
```

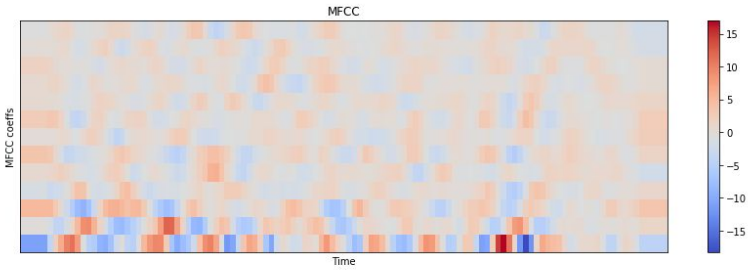
```
In [11]: # Trim the silence voice

         aa , bb = librosa.effects.trim(samples, top_db=30)
         aa, bb
```

```
Out[11]: (array([-0.00037444, -0.00035654, -0.0005744 , ..., -0.0013954 ,
                 -0.00090462, -0.00053319], dtype=float32), array([29184, 88576]))
```

```
In [12]: # Plotting Mel Power Spectrogram

         S = librosa.feature.melspectrogram(aa, sr=sample_rate, n_mels=128)

         # Convert to log scale (dB). We'll use the peak power (max) as reference.
         log_S = librosa.power_to_db(S, ref=np.max)

         plt.figure(figsize=(12, 4))
         librosa.display.specshow(log_S, sr=sample_rate, x_axis='time', y_axis='mel')
         plt.title('Mel power spectrogram ')
         plt.colorbar(format='%+02.0f dB')
         plt.tight_layout()
```




```
In [13]: # PLotting MFCC

         mfcc = librosa.feature.mfcc(S=log_S, n_mfcc=13)

         # Let's pad on the first and second deltas while we're at it

         delta2_mfcc = librosa.feature.delta(mfcc, order=2)

         plt.figure(figsize=(12, 4))
         librosa.display.specshow(delta2_mfcc)
         plt.ylabel('MFCC coeffs')
         plt.xlabel('Time')
         plt.title('MFCC')
         plt.colorbar()
         plt.tight_layout()
```




```
In [14]: # Original Sound
         ipd.Audio(samples, rate=sample_rate)
```

```
Out[14]: ▶ 0:00 / 0:05
```

```
In [15]: # Silence trimmed Sound by librosa.effects.trim()
         ipd.Audio(aa, rate=sample_rate)
```

```
Out[15]: ▶ 0:00 / 0:02
```

```
In [16]: # Silence trimmed Sound by manuel trimming
         samples_cut = samples[10000:-12500]
         ipd.Audio(samples_cut, rate=sample_rate)
```

```
Out[16]: ▶ 0:00 / 0:04
```

IV. Defining the truth label

In [17]:
```
# 2 class: Positive & Negative

# Positive: Calm, Happy
# Negative: Angry, Fearful, Sad

label2_list = []
for i in range(len(data_df)):
    if data_df.emotion[i] == 2: # Calm
        lb = "_positive"
    elif data_df.emotion[i] == 3: # Happy
        lb = "_positive"
    elif data_df.emotion[i] == 4: # Sad
        lb = "_negative"
    elif data_df.emotion[i] == 5: # Angry
        lb = "_negative"
    elif data_df.emotion[i] == 6: # Fearful
        lb = "_negative"
    else:
        lb = "_none"

    # Add gender to the label
    label2_list.append(data_df.gender[i] + lb)

len(label2_list)
```

Out[17]: 2452

In [18]:
```
#3 class: Positive, Neutral & Negative

# Positive:  Happy
# Negative: Angry, Fearful, Sad
# Neutral: Calm, Neutral

label3_list = []
for i in range(len(data_df)):
    if data_df.emotion[i] == 1: # Neutral
        lb = "_neutral"
    elif data_df.emotion[i] == 2: # Calm
        lb = "_neutral"
    elif data_df.emotion[i] == 3: # Happy
        lb = "_positive"
    elif data_df.emotion[i] == 4: # Sad
        lb = "_negative"
    elif data_df.emotion[i] == 5: # Angry
        lb = "_negative"
    elif data_df.emotion[i] == 6: # Fearful
        lb = "_negative"
    else:
        lb = "_none"

    # Add gender to the label
    label3_list.append(data_df.gender[i] + lb)

len(label3_list)
```

Out[18]: 2452

In [19]:
```
# 5 class: angry, calm, sad, happy & fearful
label5_list = []
for i in range(len(data_df)):
    if data_df.emotion[i] == 2:
        lb = "_calm"
    elif data_df.emotion[i] == 3:
        lb = "_happy"
    elif data_df.emotion[i] == 4:
        lb = "_sad"
    elif data_df.emotion[i] == 5:
        lb = "_angry"
    elif data_df.emotion[i] == 6:
        lb = "_fearful"
    else:
        lb = "_none"

    # Add gender to the label
    label5_list.append(data_df.gender[i] + lb)

len(label5_list)
```

Out[19]: 2452

In [20]:
```
# All class

label8_list = []
for i in range(len(data_df)):
    if data_df.emotion[i] == 1:
        lb = "_neutral"
    elif data_df.emotion[i] == 2:
        lb = "_calm"
    elif data_df.emotion[i] == 3:
        lb = "_happy"
    elif data_df.emotion[i] == 4:
        lb = "_sad"
    elif data_df.emotion[i] == 5:
        lb = "_angry"
    elif data_df.emotion[i] == 6:
        lb = "_fearful"
    elif data_df.emotion[i] == 7:
        lb = "_disgust"
    elif data_df.emotion[i] == 8:
        lb = "_surprised"
    else:
        lb = "_none"

    # Add gender to the label
    label8_list.append(data_df.gender[i]  + lb)

len(label8_list)
```

Out[20]:
```
2452
```

In [21]:
```
# Selecting the label set we want by commenting the unwanteds.

data_df['label'] = label2_list
# data_df['label'] = label3_list
# data_df['label'] = label5_list
# data_df['label'] = label8_list
data_df.head()
```

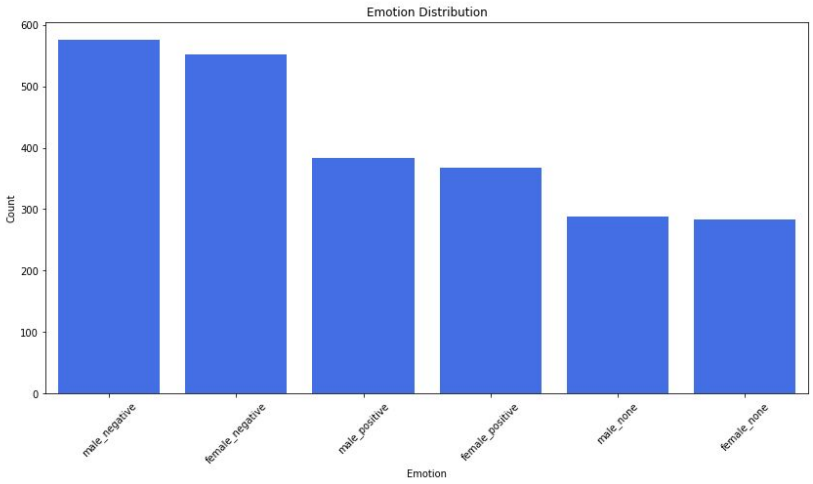
Out[21]:

	path	source	actor	gender	intensity	statement	repetition	emotion	label
0	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	0	0	1	male_none
1	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	0	1	1	male_none
2	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	1	0	1	male_none
3	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	1	1	1	male_none
4	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	0	0	2	male_positive

In [22]:
```
print (data_df.label.value_counts().keys())
```

```
Index(['male_negative', 'female_negative', 'male_positive', 'female_positive',
       'male_none', 'female_none'],
      dtype='object')
```

In [24]:
```
a = data_df.label.value_counts()
plot_emotion_dist(a, "#2962FF", "Emotion Distribution")
```



In [23]:
```
# Plotting the emotion distribution

def plot_emotion_dist(dist, color_code='#C2185B', title="Plot"):
    """
    To plot the data distributioin by class.
    Arg:
      dist: pandas series of label count.
    """
    tmp_df = pd.DataFrame()
    tmp_df['Emotion'] = list(dist.keys())
    tmp_df['Count'] = list(dist)
    fig, ax = plt.subplots(figsize=(14, 7))
    ax = sns.barplot(x="Emotion", y='Count', color=color_code, data=tmp_df)
    ax.set_title(title)
    ax.set_xticklabels(ax.get_xticklabels(),rotation=45)
```

V. Data Splitting

In [25]:
```
# Female Data Set

## Uncomment all below to use Female set

# data2_df = data_df.copy()
# data2_df = data2_df[data2_df.label != "male_none"]
# data2_df = data2_df[data2_df.label != "female_none"]
# data2_df = data2_df[data2_df.label != "male_happy"]
# data2_df = data2_df[data2_df.label != "male_angry"]
# data2_df = data2_df[data2_df.label != "male_sad"]
# data2_df = data2_df[data2_df.label != "male_fearful"]
# data2_df = data2_df[data2_df.label != "male_calm"]
# data2_df = data2_df[data2_df.label != "male_positive"]
# data2_df = data2_df[data2_df.label != "male_negative"].reset_index(drop=True)

# tmp1 = data2_df[data2_df.actor == 22]
# tmp2 = data2_df[data2_df.actor == 24]
# data3_df = pd.concat([tmp1, tmp2],ignore_index=True).reset_index(drop=True)
# data2_df = data2_df[data2_df.actor != 22]
# data2_df = data2_df[data2_df.actor != 24].reset_index(drop=True)
# print (len(data2_df))
# data2_df.head()
```

In [26]:
```
# Male Data Set

## Uncomment all below to use Male set

data2_df = data_df.copy()
data2_df = data2_df[data2_df.label != "male_none"]
data2_df = data2_df[data2_df.label != "female_none"].reset_index(drop=True)
data2_df = data2_df[data2_df.label != "female_neutral"]
data2_df = data2_df[data2_df.label != "female_happy"]
data2_df = data2_df[data2_df.label != "female_angry"]
data2_df = data2_df[data2_df.label != "female_sad"]
data2_df = data2_df[data2_df.label != "female_fearful"]
data2_df = data2_df[data2_df.label != "female_calm"]
data2_df = data2_df[data2_df.label != "female_positive"]
data2_df = data2_df[data2_df.label != "female_negative"].reset_index(drop=True)

tmp1 = data2_df[data2_df.actor == 21]
tmp2 = data2_df[data2_df.actor == 22]
tmp3 = data2_df[data2_df.actor == 23]
tmp4 = data2_df[data2_df.actor == 24]
data3_df = pd.concat([tmp1, tmp3],ignore_index=True).reset_index(drop=True)
data2_df = data2_df[data2_df.actor != 21]
data2_df = data2_df[data2_df.actor != 22]
data2_df = data2_df[data2_df.actor != 23].reset_index(drop=True)
data2_df = data2_df[data2_df.actor != 24].reset_index(drop=True)
print (len(data2_df))
data2_df.head()
```

```
800
```

Out[26]:

	path	source	actor	gender	intensity	statement	repetition	emotion	label
0	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	0	0	2	male_positive
1	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	0	1	2	male_positive
2	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	1	0	2	male_positive
3	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	0	1	1	2	male_positive
4	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	1	male	1	0	0	2	male_positive

In [27]:
```
print (len(data3_df))
data3_df.head()
```

```
160
```

Out[27]:

	path	source	actor	gender	intensity	statement	repetition	emotion	label
0	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	21	male	0	0	0	2	male_positive
1	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	21	male	0	0	1	2	male_positive
2	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	21	male	0	1	0	2	male_positive
3	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	21	male	0	1	1	2	male_positive
4	C://Users//pranav//Desktop//SER//xdata//Actor_...	1	21	male	1	0	0	2	male_positive

VI. Getting the features of audio files using librosa

In [28]:
```
data = pd.DataFrame(columns=['feature'])
for i in tqdm(range(len(data2_df))):
    X, sample_rate = librosa.load(data2_df.path[i], res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
#     X = X[10000:90000]
    sample_rate = np.array(sample_rate)
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
    feature = mfccs
    data.loc[i] = [feature]
```

```
100%|██████████| 800/800 [01:28<00:00,  9.44it/s]
```

In [29]:
```
data.head()
```

Out[29]:

	feature
0	[-70.2677641610773, -70.2677641610773, -70.267...
1	[-67.55739512198222, -67.55739512198222, -67.5...
2	[-69.67328949566406, -69.69331084873151, -69.6...
3	[-69.05139995492158, -69.05139995492158, -69.0...
4	[-73.8413701111492, -73.8413701111492, -73.841...

In [30]:
```
df3 = pd.DataFrame(data['feature'].values.tolist())
labels = data2_df.label
```

In [31]:
```
df3.head()
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9	...	249	250	251
0	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	...	-70.267764	-70.267764	-69.957707
1	-67.557395	-67.557395	-67.557395	-67.557395	-67.557395	-67.557395	-65.239801	-65.536197	-67.557395	-67.557395	...	-67.557395	-67.557395	-67.557395
2	-69.673289	-69.693311	-69.693311	-69.693311	-69.693311	-69.693311	-69.693311	-69.620774	-69.693311	-68.906572	...	-69.693311	-69.693311	-69.693311
3	-69.051400	-69.051400	-69.051400	-69.051400	-69.051400	-68.754863	-69.051400	-69.051400	-69.051400	-68.359101	...	-65.446950	-68.552088	-69.051400
4	-73.841370	-73.841370	-73.841370	-73.719655	-73.841370	-73.841370	-73.841370	-73.303635	-72.806811	-73.841370	...	-73.841370	-73.841370	-73.841370

5 rows × 259 columns

In [32]:
```
newdf = pd.concat([df3,labels], axis=1)
```

In [33]:
```
rnewdf = newdf.rename(index=str, columns={"0": "label"})
len(rnewdf)
```

Out[33]:
```
800
```

In [34]:
```
rnewdf.head(10)
```

Out[34]:

	0	1	2	3	4	5	6	7	8	9	...	250	251	252
0	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	...	-70.267764	-69.957707	-68.377602
1	-67.557395	-67.557395	-67.557395	-67.557395	-67.557395	-67.557395	-65.239801	-65.536197	-67.557395	-67.557395	...	-67.557395	-67.557395	-67.557395
2	-69.673289	-69.693311	-69.693311	-69.693311	-69.693311	-69.693311	-69.693311	-69.620774	-69.693311	-68.906572	...	-69.693311	-69.693311	-69.693311
3	-69.051400	-69.051400	-69.051400	-69.051400	-69.051400	-68.754863	-69.051400	-69.051400	-69.051400	-68.359101	...	-68.552088	-69.051400	-69.051400
4	-73.841370	-73.841370	-73.841370	-73.719655	-73.841370	-73.841370	-73.841370	-73.303635	-72.806811	-73.841370	...	-73.841370	-73.841370	-73.841370
5	-69.243253	-69.243253	-69.243253	-69.243253	-68.901972	-67.982999	-68.089201	-67.897329	-65.258010	-67.170980	...	-57.185978	-61.188731	-67.108389
6	-73.254968	-73.254968	-73.254968	-73.254968	-68.774422	-69.380388	-73.254968	-73.254968	-73.254968	-73.254968	...	-50.884085	-55.666730	-54.600013
7	-70.746514	-70.746514	-70.025286	-69.131263	-70.746514	-70.746514	-70.746514	-70.746514	-70.746514	-70.746514	...	-70.746514	-70.746514	-70.079249
8	-63.311078	-63.072484	-63.412433	-63.796762	-63.581991	-58.921211	-57.955046	-61.224968	-63.782931	-63.796762	...	-63.740612	-62.410257	-62.489080
9	-60.369038	-60.083715	-60.978925	-60.952456	-60.982486	-60.983948	-60.981255	-60.981255	-60.981255	-60.249618	...	-60.981255	-60.981255	-60.981255

10 rows × 260 columns

In [35]:
```
rnewdf.isnull().sum().sum()
```

Out[35]: 2284

In [36]:
```
rnewdf = rnewdf.fillna(0)
rnewdf.head()
```

Out[36]:

	0	1	2	3	4	5	6	7	8	9	...	250	251	252
0	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	...	-70.267764	-69.957707	-68.377602
1	-67.557395	-67.557395	-67.557395	-67.557395	-67.557395	-67.557395	-65.239801	-65.536197	-67.557395	-67.557395	...	-67.557395	-67.557395	-67.557395
2	-69.673289	-69.693311	-69.693311	-69.693311	-69.693311	-69.693311	-69.693311	-69.620774	-69.693311	-68.906572	...	-69.693311	-69.693311	-69.693311
3	-69.051400	-69.051400	-69.051400	-69.051400	-69.051400	-68.754863	-69.051400	-69.051400	-69.051400	-68.359101	...	-68.552088	-69.051400	-69.051400
4	-73.841370	-73.841370	-73.841370	-73.719655	-73.841370	-73.841370	-73.841370	-73.303635	-72.806811	-73.841370	...	-73.841370	-73.841370	-73.841370

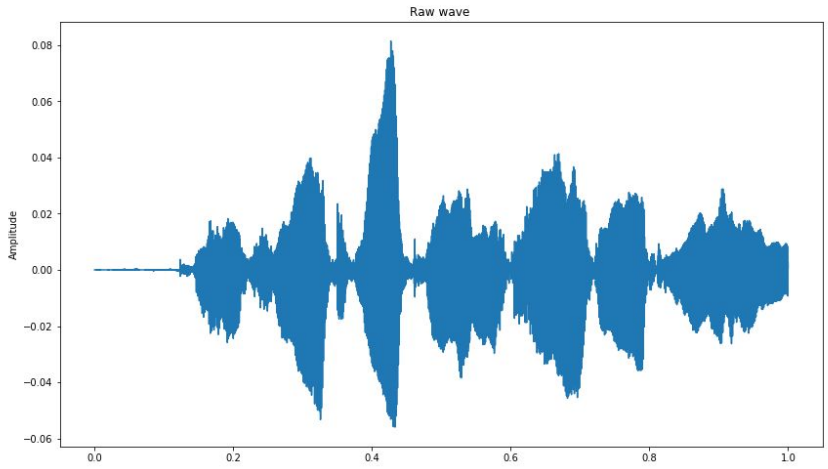
5 rows × 260 columns

VII. Data Augmentation

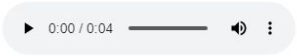
In [37]:
```
def plot_time_series(data):
    """
    Plot the Audio Frequency.
    """
    fig = plt.figure(figsize=(14, 8))
    plt.title('Raw wave ')
    plt.ylabel('Amplitude')
    plt.plot(np.linspace(0, 1, len(data)), data)
    plt.show()


def noise(data):
    """
    Adding White Noise.
    """
    # we can take any distribution from https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.random.html
    noise_amp = 0.005*np.random.uniform()*np.amax(data)
    data = data.astype('float64') + noise_amp * np.random.normal(size=data.shape[0])
    return data

def shift(data):
    """
    Random Shifting.
    """
    s_range = int(np.random.uniform(low=-5, high = 5)*500)
    return np.roll(data, s_range)
```

```python
def stretch(data, rate=0.8):
    """
    Streching the Sound.
    """
    data = librosa.effects.time_stretch(data, rate)
    return data

def pitch(data, sample_rate):
    """
    Pitch Tuning.
    """
    bins_per_octave = 12
    pitch_pm = 2
    pitch_change =  pitch_pm * 2*(np.random.uniform())
    data = librosa.effects.pitch_shift(data.astype('float64'),
                                       sample_rate, n_steps=pitch_change,
                                       bins_per_octave=bins_per_octave)
    return data

def dyn_change(data):
    """
    Random Value Change.
    """
    dyn_change = np.random.uniform(low=1.5,high=3)
    return (data * dyn_change)

def speedNpitch(data):
    """
    peed and Pitch Tuning.
    """
    # we can change low and high here
    length_change = np.random.uniform(low=0.8, high = 1)
    speed_fac = 1.0  / length_change
    tmp = np.interp(np.arange(0,len(data),speed_fac),np.arange(0,len(data)),data)
    minlen = min(data.shape[0], tmp.shape[0])
    data *= 0
    data[0:minlen] = tmp[0:minlen]
    return data
```

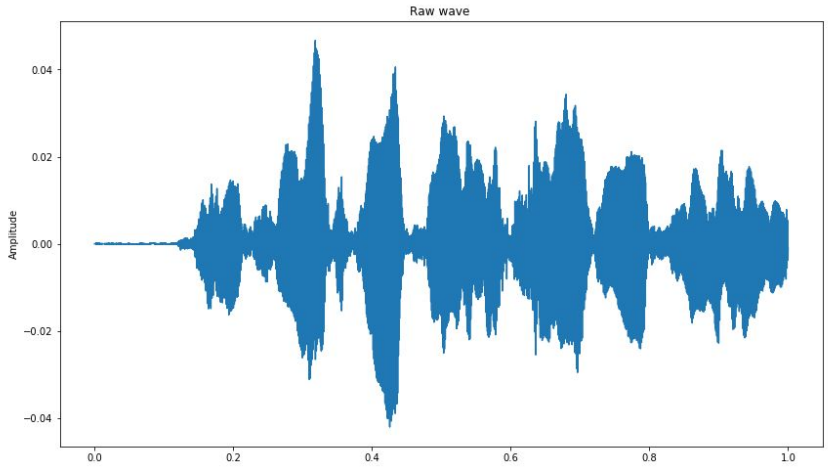
In [38]:
```python
X, sample_rate = librosa.load(data2_df.path[216], res_type='kaiser_fast',duration=4,sr=22050*2,offset=0.5)
plot_time_series(X)
ipd.Audio(X, rate=sample_rate)
```



Out[38]:



In [39]:
```
x = pitch(X, sample_rate)
plot_time_series(x)
ipd.Audio(x, rate=sample_rate)
```



Out[39]:

In [40]:
```
# Augmentation Method 1

syn_data1 = pd.DataFrame(columns=['feature', 'label'])
for i in tqdm(range(len(data2_df))):
    X, sample_rate = librosa.load(data2_df.path[i], res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
    if data2_df.label[i]:
#     if data2_df.label[i] == "male_positive":
        X = noise(X)
        sample_rate = np.array(sample_rate)
        mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
        feature = mfccs
        a = random.uniform(0, 1)
        syn_data1.loc[i] = [feature, data2_df.label[i]]
```

```
100%|██████████| 800/800 [01:38<00:00,  8.15it/s]
```

In [41]:
```
# Augmentation Method 2

syn_data2 = pd.DataFrame(columns=['feature', 'label'])
for i in tqdm(range(len(data2_df))):
    X, sample_rate = librosa.load(data2_df.path[i], res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
    if data2_df.label[i]:
#     if data2_df.label[i] == "male_positive":
        X = pitch(X, sample_rate)
        sample_rate = np.array(sample_rate)
        mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
        feature = mfccs
        a = random.uniform(0, 1)
        syn_data2.loc[i] = [feature, data2_df.label[i]]
```

```
100%|██████████| 800/800 [08:31<00:00,  1.62it/s]
```

In [42]:
```
len(syn_data1), len(syn_data2)
```

Out[42]:
```
(800, 800)
```

In [43]:
```
syn_data1 = syn_data1.reset_index(drop=True)
syn_data2 = syn_data2.reset_index(drop=True)
```

In [44]:
```
df4 = pd.DataFrame(syn_data1['feature'].values.tolist())
labels4 = syn_data1.label
syndf1 = pd.concat([df4,labels4], axis=1)
syndf1 = syndf1.rename(index=str, columns={"0": "label"})
syndf1 = syndf1.fillna(0)
len(syndf1)
```

Out[44]: 800

In [45]:
```
syndf1.head()
```

Out[45]:

	0	1	2	3	4	5	6	7	8	9	...	250	251	252
0	-61.920078	-61.220244	-62.322067	-61.775916	-61.688969	-62.836432	-63.249473	-62.082875	-61.559745	-60.062484	...	-58.546197	-59.756362	-61.582480
1	-57.721158	-55.123507	-53.067941	-55.741827	-55.857400	-54.822464	-55.950613	-55.404070	-55.087911	-54.761412	...	-55.526578	-54.617210	-53.443551
2	-62.466070	-60.134119	-61.538021	-62.081440	-63.256823	-61.795986	-61.136131	-62.355826	-60.940998	-61.795114	...	-59.865914	-60.895722	-63.057329
3	-68.897608	-68.868628	-67.272517	-67.746347	-67.904868	-67.022255	-67.692426	-68.523103	-68.916927	-68.374147	...	-67.723046	-68.452296	-68.393548
4	-68.851856	-67.490619	-67.247325	-65.555171	-65.570013	-65.738487	-68.317115	-65.936454	-63.993811	-63.210834	...	-66.262867	-67.594036	-66.693545

5 rows × 260 columns

In [46]:
```
df4 = pd.DataFrame(syn_data2['feature'].values.tolist())
labels4 = syn_data2.label
syndf2 = pd.concat([df4,labels4], axis=1)
syndf2 = syndf2.rename(index=str, columns={"0": "label"})
syndf2 = syndf2.fillna(0)
len(syndf2)
```

Out[46]: 800

In [47]:
```
syndf2.head()
```

Out[47]:

	0	1	2	3	4	5	6	7	8	9	...	250	251	252
0	-72.535069	-72.535069	-72.535069	-72.535069	-72.535069	-72.535069	-72.535069	-72.535069	-72.535069	-72.535069	...	-72.535069	-72.535069	-72.326269
1	-71.578808	-71.480058	-71.306878	-71.578808	-71.578808	-70.161384	-69.542567	-71.237882	-71.578808	-71.578808	...	-71.578808	-71.578808	-71.578808
2	-71.605278	-71.565580	-72.316263	-72.316263	-72.316263	-72.316263	-72.316263	-72.316263	-72.233252	-71.146702	...	-72.316263	-72.316263	-72.316263
3	-70.956062	-70.956062	-70.956062	-70.956062	-70.956062	-70.948397	-70.956062	-70.956062	-70.956062	-70.956062	...	-69.063367	-70.956062	-70.956062
4	-74.258629	-74.258629	-74.258629	-74.258629	-74.258629	-74.258629	-74.258629	-74.258629	-73.737727	-74.258629	...	-74.258629	-74.258629	-74.258629

5 rows × 260 columns

In [48]:
```
# Combining the Augmented data with original
combined_df = pd.concat([rnewdf, syndf1, syndf2], ignore_index=True)
combined_df = combined_df.fillna(0)
combined_df.head()
```

Out[48]:

	0	1	2	3	4	5	6	7	8	9	...	250	251	252
0	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	-70.267764	...	-70.267764	-69.957707	-68.377602
1	-67.557395	-67.557395	-67.557395	-67.557395	-67.557395	-67.557395	-65.239801	-65.536197	-67.557395	-67.557395	...	-67.557395	-67.557395	-67.557395
2	-69.673289	-69.693311	-69.693311	-69.693311	-69.693311	-69.693311	-69.693311	-69.620774	-69.693311	-68.906572	...	-69.693311	-69.693311	-69.693311
3	-69.051400	-69.051400	-69.051400	-69.051400	-69.051400	-68.754863	-69.051400	-69.051400	-69.051400	-68.359101	...	-68.552088	-69.051400	-69.051400
4	-73.841370	-73.841370	-73.841370	-73.719655	-73.841370	-73.841370	-73.841370	-73.303635	-72.806811	-73.841370	...	-73.841370	-73.841370	-73.841370

5 rows × 260 columns

In [49]:
```
#  Stratified Shuffle Split

X = combined_df.drop(['label'], axis=1)
y = combined_df.label
xxx = StratifiedShuffleSplit(1, test_size=0.2, random_state=12)
for train_index, test_index in xxx.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

In [50]:
```
y_train.value_counts()
```

Out[50]:
```
male_negative    1152
male_positive     768
Name: label, dtype: int64
```

In [51]:
```
y_test.value_counts()
```

Out[51]:
```
male_negative    288
male_positive    192
Name: label, dtype: int64
```

In [52]:
```
X_train.isna().sum().sum()
```

Out[52]:
```
0
```

In [53]:
```
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))
```

In [54]:
```
y_train
```

Out[54]:
```
array([[1., 0.],
       [0., 1.],
       [1., 0.],
       ...,
       [1., 0.],
       [0., 1.],
       [0., 1.]], dtype=float32)
```

In [55]:
```
X_train.shape
```

Out[55]:
```
(1920, 259)
```

VIII. Changing dimension for CNN model

In [56]:
```
x_traincnn = np.expand_dims(X_train, axis=2)
x_testcnn = np.expand_dims(X_test, axis=2)
```

In [57]:
```
# Set up Keras util functions

from keras import backend as K

def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def fscore(y_true, y_pred):
    if K.sum(K.round(K.clip(y_true, 0, 1))) == 0:
        return 0

    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    f_score = 2 * (p * r) / (p + r + K.epsilon())
    return f_score

def get_lr_metric(optimizer):
    def lr(y_true, y_pred):
        return optimizer.lr
    return lr
```

In [58]:
```
# New model
model = Sequential()
model.add(Conv1D(256, 8, padding='same',input_shape=(X_train.shape[1],1)))
model.add(Activation('relu'))
model.add(Conv1D(256, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Flatten())


# Edit according to target class no.
model.add(Dense(2))
model.add(Activation('softmax'))
opt = keras.optimizers.SGD(lr=0.0001, momentum=0.0, decay=0.0, nesterov=False)
```

```
WARNING:tensorflow:From F:\New folder\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [59]:
```
# Original Model

# model = Sequential()
# model.add(Conv1D(256, 5,padding='same', input_shape=(X_train.shape[1],1)))
# model.add(Activation('relu'))
# model.add(Conv1D(128, 5,padding='same'))
# model.add(Activation('relu'))
# model.add(Dropout(0.1))
# model.add(MaxPooling1D(pool_size=(8)))
# model.add(Conv1D(128, 5,padding='same',))
# model.add(Activation('relu'))
# model.add(Conv1D(128, 5,padding='same',))
# model.add(Activation('relu'))
# model.add(Conv1D(128, 5,padding='same',))
# model.add(Activation('relu'))
# model.add(Dropout(0.2))
# model.add(Conv1D(128, 5,padding='same',))
# model.add(Activation('relu'))
# model.add(Flatten())
# model.add(Dense(5))
# model.add(Activation('softmax'))
# opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
```

In [60]:
```
# Plotting Model Summary

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 259, 256)          2304
_________________________________________________________________
activation_1 (Activation)    (None, 259, 256)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 259, 256)          524544
_________________________________________________________________
batch_normalization_1 (Batch (None, 259, 256)          1024
_________________________________________________________________
activation_2 (Activation)    (None, 259, 256)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 259, 256)          0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 32, 256)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 32, 128)           262272
_________________________________________________________________
activation_3 (Activation)    (None, 32, 128)           0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 32, 128)           131200
_________________________________________________________________
activation_4 (Activation)    (None, 32, 128)           0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 32, 128)           131200
_________________________________________________________________
activation_5 (Activation)    (None, 32, 128)           0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 32, 128)           131200
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 128)           512
_________________________________________________________________
activation_6 (Activation)    (None, 32, 128)           0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 128)           0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 4, 128)            0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 4, 64)             65600
_________________________________________________________________
activation_7 (Activation)    (None, 4, 64)             0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 4, 64)             32832
_________________________________________________________________
activation_8 (Activation)    (None, 4, 64)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 514
_________________________________________________________________
activation_9 (Activation)    (None, 2)                 0
=================================================================
Total params: 1,283,202
Trainable params: 1,282,434
Non-trainable params: 768
_________________________________________________________________
```

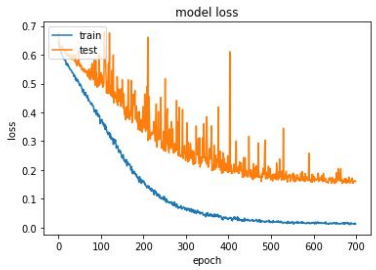
In [61]:
```
# Compile my model

model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy', fscore])
```

IX. Removed the whole training part for avoiding unnecessary long epochs list

In [63]:
```
# Model Training

lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=20, min_lr=0.000001)
# set the model name accordingly.
mcp_save = ModelCheckpoint('C://Users//pranav//Desktop//SER//saved_models//Emotion_Voice_Detection_Model.h5', save_best_only=Tru
cnnhistory=model.fit(x_traincnn, y_train, batch_size=16, epochs=700,
                     validation_data=(x_testcnn, y_test), callbacks=[mcp_save, lr_reduce])
```

```
Train on 1920 samples, validate on 480 samples
Epoch 1/700
1920/1920 [==============================] - ETA: 1:06 - loss: 0.6422 - accuracy: 0.6250 - fscore: 0.625 - ETA: 1:04 - loss: 0.
6326 - accuracy: 0.6875 - fscore: 0.687 - ETA: 1:03 - loss: 0.6491 - accuracy: 0.6458 - fscore: 0.645 - ETA: 1:04 - loss: 0.645
2 - accuracy: 0.6719 - fscore: 0.671 - ETA: 1:03 - loss: 0.6432 - accuracy: 0.6625 - fscore: 0.662 - ETA: 1:03 - loss: 0.6540 -
accuracy: 0.6458 - fscore: 0.645 - ETA: 1:03 - loss: 0.6550 - accuracy: 0.6429 - fscore: 0.642 - ETA: 1:02 - loss: 0.6521 - acc
uracy: 0.6328 - fscore: 0.632 - ETA: 1:02 - loss: 0.6560 - accuracy: 0.6181 - fscore: 0.618 - ETA: 1:02 - loss: 0.6456 - accura
cy: 0.6375 - fscore: 0.637 - ETA: 1:02 - loss: 0.6481 - accuracy: 0.6364 - fscore: 0.636 - ETA: 1:01 - loss: 0.6501 - accuracy:
0.6250 - fscore: 0.625 - ETA: 1:00 - loss: 0.6504 - accuracy: 0.6250 - fscore: 0.625 - ETA: 59s - loss: 0.6466 - accuracy: 0.63
84 - fscore: 0.638 - ETA: 58s - loss: 0.6455 - accuracy: 0.6375 - fscore: 0.63 - ETA: 58s - loss: 0.6479 - accuracy: 0.6328 - f
```

In [64]:
```
# Plotting the Train Valid Loss Graph

plt.plot(cnnhistory.history['loss'])
plt.plot(cnnhistory.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Saving the model

In [65]:
```
# Saving the model.json

import json
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

Loading the model

In [70]:
```
# Loading json and creating model
from keras.models import model_from_json
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# Load weights into new model
loaded_model.load_weights("C://Users//pranav//Desktop//SER//Emotion_Voice_Detection_CNNModel.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
score = loaded_model.evaluate(x_testcnn, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
Loaded model from disk
accuracy: 75.21%
```

X. Predicting emotions on the test data

In [71]:
```
len(data3_df)
```

Out[71]:
```
160
```

In [72]:
```
data_test = pd.DataFrame(columns=['feature'])
for i in tqdm(range(len(data3_df))):
    X, sample_rate = librosa.load(data3_df.path[i], res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
#     X = X[10000:90000]
    sample_rate = np.array(sample_rate)
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
    feature = mfccs
    data_test.loc[i] = [feature]

test_valid = pd.DataFrame(data_test['feature'].values.tolist())
test_valid = np.array(test_valid)
test_valid_lb = np.array(data3_df.label)
lb = LabelEncoder()
test_valid_lb = np_utils.to_categorical(lb.fit_transform(test_valid_lb))
test_valid = np.expand_dims(test_valid, axis=2)
```

```
100%|██████████| 160/160 [00:30<00:00,  5.31it/s]
```

In [73]:
```
preds = loaded_model.predict(test_valid,
                             batch_size=16,
                             verbose=1)
```

```
160/160 [==============================] - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - 3s 20ms/step
```

In [74]:
```
preds
```

Out[74]:
```
array([[4.04060371e-02, 9.59593952e-01],
       [5.42703457e-02, 9.45729673e-01],
       [3.77489775e-01, 6.22510254e-01],
       [7.09618628e-03, 9.92903769e-01],
       [5.07596493e-01, 4.92403537e-01],
       [3.10853403e-03, 9.96891439e-01],
       [1.18919031e-03, 9.98810768e-01],
       [2.75274063e-03, 9.97247279e-01],
       [6.36962950e-02, 9.36303735e-01],
       [1.07676268e-01, 8.92323732e-01],
       [7.76831508e-01, 2.23168522e-01],
       [1.02844983e-01, 8.97155046e-01],
       [3.51285040e-01, 6.48714900e-01],
       [2.53522824e-02, 9.74647760e-01],
       [8.77780259e-01, 1.22219764e-01],
       [5.54001808e-01, 4.45998162e-01],
       [1.13658719e-02, 9.88634109e-01],
       [8.69896114e-02, 9.13010418e-01],
       [6.40829501e-04, 9.99359190e-01],
       [1.93699834e-05, 9.99980688e-01],
       [7.19594285e-02, 9.28040564e-01],
       [5.22732874e-03, 9.94772732e-01],
       [2.40910216e-03, 9.97590899e-01],
       [2.45080399e-03, 9.97549236e-01],
       [4.43860143e-02, 9.55613971e-01],
       [5.24361193e-01, 4.75638807e-01],
       [8.95550847e-01, 1.04449153e-01],
       [9.44449723e-01, 5.55503033e-02],
       [1.64212435e-01, 8.35787594e-01],
       [8.75178814e-01, 1.24821171e-01],
       [6.47591591e-01, 3.52408409e-01],
```

In [75]:
```
preds1=preds.argmax(axis=1)
```

In [76]:
```
preds1
```

Out[76]:
```
array([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 1], dtype=int64)
```

In [77]:
```
abc = preds1.astype(int).flatten()
```

In [78]:
```
predictions = (lb.inverse_transform((abc)))
```

In [79]:
```
preddf = pd.DataFrame({'predictedvalues': predictions})
preddf[:10]
```

Out[79]:

	predictedvalues
0	male_positive
1	male_positive
2	male_positive
3	male_positive
4	male_negative
5	male_positive
6	male_positive
7	male_positive
8	male_positive
9	male_positive

In [80]:
```
actual=test_valid_lb.argmax(axis=1)
abc123 = actual.astype(int).flatten()
actualvalues = (lb.inverse_transform((abc123)))
```

In [81]:
```
actualdf = pd.DataFrame({'actualvalues': actualvalues})
actualdf[:10]
```

Out[81]:

	actualvalues
0	male_positive
1	male_positive
2	male_positive
3	male_positive
4	male_positive
5	male_positive
6	male_positive
7	male_positive
8	male_positive
9	male_positive

In [82]:
```
finaldf = actualdf.join(preddf)
```

Actual v/s Predicted emotions

In [83]:
```
finaldf[20:40]
```

Out[83]:

	actualvalues	predictedvalues
20	male_negative	male_positive
21	male_negative	male_positive
22	male_negative	male_positive
23	male_negative	male_positive
24	male_negative	male_positive
25	male_negative	male_negative
26	male_negative	male_negative
27	male_negative	male_negative
28	male_negative	male_positive
29	male_negative	male_negative
30	male_negative	male_negative

In [84]:
```
finaldf.groupby('actualvalues').count()
```

Out[84]:

actualvalues	predictedvalues
male_negative	96
male_positive	64

In [85]:
```
finaldf.groupby('predictedvalues').count()
```

Out[85]:

predictedvalues	actualvalues
male_negative	60
male_positive	100

In [86]:
```
finaldf.to_csv('Predictions.csv', index=False)
```

In [87]:
```
def print_confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsize=14):
    """Prints a confusion matrix, as returned by sklearn.metrics.confusion_matrix, as a heatmap.

    Arguments
    ---------
    confusion_matrix: numpy.ndarray
        The numpy.ndarray object returned from a call to sklearn.metrics.confusion_matrix.
        Similarly constructed ndarrays can also be used.
    class_names: list
        An ordered list of class names, in the order they index the given confusion matrix.
    figsize: tuple
        A 2-long tuple, the first value determining the horizontal size of the ouputted figure,
        the second determining the vertical size. Defaults to (10,7).
    fontsize: int
        Font size for axes labels. Defaults to 14.

    Returns
    -------
    matplotlib.figure.Figure
        The resulting confusion matrix figure
    """
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")

    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [88]:
```
from sklearn.metrics import accuracy_score
y_true = finaldf.actualvalues
y_pred = finaldf.predictedvalues
accuracy_score(y_true, y_pred)*100
```

Out[88]:
```
61.25000000000001
```

In [89]:
```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred, average='macro') *100
```

Out[89]:
```
61.225766103814884
```

In [90]:
```
from sklearn.metrics import confusion_matrix
c = confusion_matrix(y_true, y_pred)
c
```

Out[90]:
```
array([[47, 49],
       [13, 51]], dtype=int64)
```

In [91]:
```
# Visualize Confusion Matrix

# class_names = ['male_angry', 'male_calm', 'male_fearful', 'male_happy', 'male_sad']
# class_names = ['female_angry', 'female_calm', 'female_fearful', 'female_happy', 'female_sad']
# class_names = ['male_negative', 'male_neutral', 'male_positive']
class_names = ['male_negative', 'male_positive']
# class_names = ['female_angry', 'female_calm', 'female_fearful', 'female_happy', 'female_sad', 'male_angry', 'male_calm', 'male
```

```
# class_names = ['male_angry', 'male_calm', 'male_fearful', 'male_happy', 'male_sad']
# class_names = ['female_angry', 'female_calm', 'female_fearful', 'female_happy', 'female_sad']
# class_names = ['male_negative', 'male_neutral', 'male_positive']
class_names = ['male_negative', 'male_positive']
# class_names = ['female_angry', 'female_calm', 'female_fearful', 'female_happy', 'female_sad', 'male_angry', 'male_calm', 'male_

print_confusion_matrix(c, class_names)
```

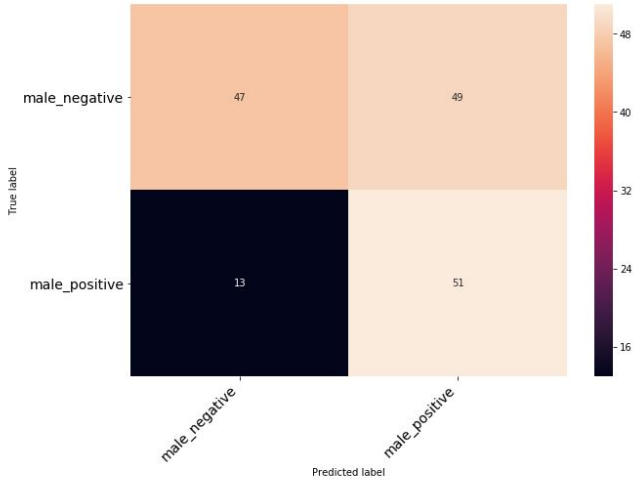


Fig: 6.1

Accuracy Score: 75.21%

Model has an accuracy of 75.21%, which is considered a good one.

Chapter 7. Conclusion and Future Scope

Conclusion

In the end, I only have time to experiment with the male data set. I re-split the data with stratified shuffle split to make sure there is no data imbalance nor data leakage problem. I tuned the model by experimenting with the male dataset since I want to simplify the model at the beginning. I also tested the by with different target label setups and augmentation methods. I found out Noise Adding and Shifting for the imbalanced data could help in achieving a better result.

Building the model was a challenging task as it involved a lot of trial and error methods, tuning etc. The model is very well trained to distinguish between male and female voices and it distinguishes with 100% accuracy. The model was tuned to detect emotions with more than **75.21% accuracy**. Accuracy can be increased by including more audio files for training.

Accuracy Score: 75.21%

Further Improvement

- I only selected the first 3 seconds to be the input data since it would reduce the dimension, the original notebook used 2.5 sec only. I would like to use the full length of the audio to do the experiment.
- Preprocess the data like cropping silence voice, normalize the length by zero padding, etc.
- Experiment the Recurrent Neural Network approach on this topic.

FUTURE SCOPE

There are a number of ways that this project could be extended. Perhaps one of the most common tools in emotion detection systems is the neural networks and hidden markov model for automatic recognition. Also the integration of other modalities such as video based or manual interaction will be investigated further.

About Me

- **GitHub:** pranavvjha/Speech-Emotion-Recognition-with-Librosa

REFERENCE : https://www.thepythoncode.com/ ,
http://neuron.arts.ryerson.ca/ravdess/?f=3,