MULTIMODAL EMOTION RECOGNITION SYSTEM

A Major-Project Report Submitted to the JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

D. Keerthana (16071A0575)

M. Keerthi (16071A0592)

P. Mahitha (16071A05A7)

V. Sumasree Reddy (16071A05B8)

Under the Guidance Of

Mrs. Bhanusree Yalamanchili

(Asst Professor, VNRVJIET)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institute, NAAC Accredited with 'A++' Grade NBA Accredited for CE, EEE, ME, ECE, CSE, EIE, IT B.Tech Courses Approved by AICTE, New Delhi, Affiliated to JNTUH Recognized as "College with Potential for Excellence" by UGC
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad 500090,TS,India.

April, 2020

VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute, NAAC Accredited with ‘A’ Grade)

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet(S.O),Hyderabad-500090.

CERTIFICATE

This is to certify that **D. Keerthana (16071A0575), M. Keerthi (16071A0592), P. Mahitha (16071A05A7), V. Sumasree Reddy (16071A05B8)** have successfully completed their project work at CSE Department of VNR VJIET, Hyderabad entitled **“Multimodal Emotion Recognition Systems**“ in partial fulfilment of the requirements for the award of B. Tech degree during the academic year 2019-2020.

Project Guide	Head of the Department
Mrs. Bhanusree Yalamanchili	**Dr. B. V. Kiranmayee**
Assistant Professor	Associate Professor and Head
CSE Department	CSE Department
VNRVJIET	VNRVJIET

DECLARATION

This is to certify that the project work entitled **"Multimodal Emotion Recognition Systems"** submitted in VNR Vignana Jyothi Institute of Engineering & Technology in partial fulfilment of requirement for the award of Bachelor of Technology in Computer Science and Engineering is a bona fide report of the work carried out by us under the guidance and supervision of Bhanusree Yalamanchili (Asst Professor), Department of CSE, VNRVJIET. To the best of our knowledge, this report has not been submitted in any form to any university or institution for the award of any degree or diploma.

D. Keerthana	**M. Keerthi**	**P. Mahitha**	**V. Sumasree Reddy**
(16071A0575)	(16071A0592)	(16071A05A7)	(16071A05B8)
IV B. Tech-CSE,	IV B.Tech-CSE,	IV B. Tech-CSE,	IV B. Tech-CSE,
VNR VJIET	VNR VJIET	VNR VJIET	VNRVJIET

ACKNOWLEDGEMENT

Behind every achievement lies an unfathomable sea of gratitude to those who activated it, without it would ever never have come into existence. To them we lay the words of gratitude imprinting within us.

We are indebted to our venerable principal **Dr. C. D. Naidu** for this unflinching devotion, which lead us to complete this project. The support, encouragement given by him and his motivation lead us to complete this project.

We express our thanks to internal guide **Mrs. Bhanusree Yalamanchili** and also Head of the department **Mrs. B. V. Kiranmayee** for having provided us a lot of facilities to undertake the project work and guide us to complete the project.

We express our sincere thanks to our faculty of the department of **Computer Science and Engineering** and the remaining members of our college **VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING**

AND TECHNOLOGY who extended their valuable support in helping us to complete the project in time.

D. Keerthana (16071A0575)

M. Keerthi (16071A0592)

P. Mahitha (16071A5A7)

V. Sumasree Reddy (16071A05B8)

ABSTRACT

The need for machines to react towards human emotions is required and is the need of the hour in the current digital era. The human-machine interaction (HCI) with responses to emotions will make them more useful and user friendly. So we developed a system which takes either audio or video as input and gives the status of emotion. We classified emotions into 7 categories like happy, sad, fearful, disgust, calm, angry and surprise. We used Ravdess dataset for speech using time distributed CNN which extracts MFCC, Mel-Scale, Spectrogram features. For video, Fer2013 dataset using CNN model with synthesized features like haar features, HOG sliding windows, HOG features and Facial landmarks are used. By applying these algorithms we have achieved an accuracy of about 96% on training and about 75% on validation when speech emotion model is used and about 89% accuracy on training and 76% on validation when face emotion model is used. Later, these modules can be integrated in interactive devices such as smartphones, smart wearables, voice assistants and other IoT devices.

Contents

CHAPTER 1
INTRODUCTION

In today's world, the rapid growth of artificial intelligence has escalated the need for better and natural interaction between humans and machines. Emotion recognition systems can be deployed into diverse applications. The information obtained from these systems is being used in fields like health, education, tourism and commerce, etc. Unimodal systems cannot provide more information about the user and to various exterior factors. This has led to the development of multimodal systems rather than unimodal systems. The ways to recognize the emotions of users are asking from a user, Voice recognition, Tracking implicit parameters, Facial expression recognition, Gesture recognition ,Vital signals, and Hybrid methods. Physiological features such as respiratory volume, skin temperature, heart rate, respiration pattern, EDA, PPG, and EMG can also be used to determine emotion. For multimodal systems to give accurate results using different types of data, fusion techniques are used. Various fusion techniques which can be used are feature-level, hybrid multimodal, decision- level, rule-based, classification-based, model-level, and estimation-based fusions. The databases accumulated by researchers for these systems can contain only text, only audio, only image, only video, audio and video, physiological data or audio, text, and video data.

CHAPTER 2

EXISTING SYSTEM AND PROPOSED SYSTEM

2.1 EXISTING SYSTEM

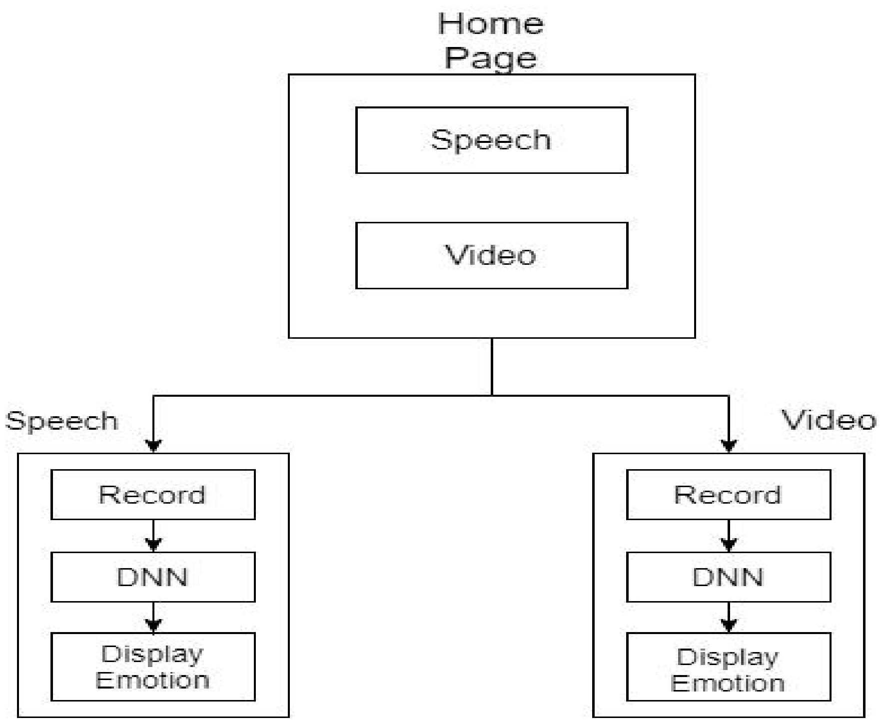
The emotion recognition systems developed till date try to percept information using data from facial expression, speech, text, gestures and other physiological features. The techniques such as signal processing, computer vision, machine learning, and speech processing are used. These techniques are classified into three categories- knowledge-based, statistical, and hybrid approaches. Knowledge based techniques use semantics and syntax of languages to determine the emotion. They can be dictionary based or corpus based. Statistical methods commonly use supervised machine learning algorithms to predict emotions and give more accurate results. Deep learning algorithms such as CNN and LSTM are also used in these methods. Hybrid approaches are a combination of knowledge based and statistical techniques. The computation in hybrid approaches is more complex which is the reason for its scarce usage.

The unimodal systems use only one type of data such as audio, video or text. These methods do not yield accurate results when there are external disturbances. The multimodal systems being used generally use audio, video and text data to extract features. The features extracted are then fused to predict emotions. While these methods give more accurate results, they need to access all types of data in order to deduce emotions. This may raise privacy concerns. If even one of the types of data is missing, the fusion is not possible and therefore results cannot be given.

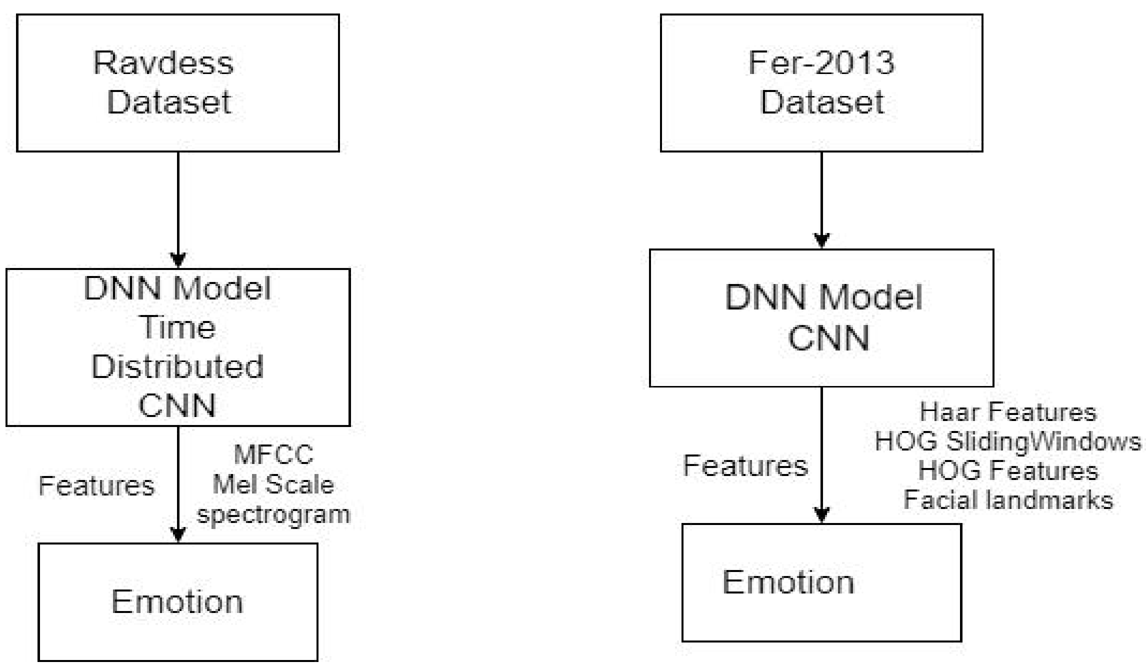
2.2 PROPOSED SOLUTION:

The proposed system tries to solve these issues in emotion recognition systems. The users are given a choice for what type of data they want to share in order to predict emotions. This resolves the privacy concerns and even if one type of data is not available, other types of data can be used. This ensures the working of the system under all conditions.

Proposed System:



Proposed Neural Network Architecture:



CHAPTER 3

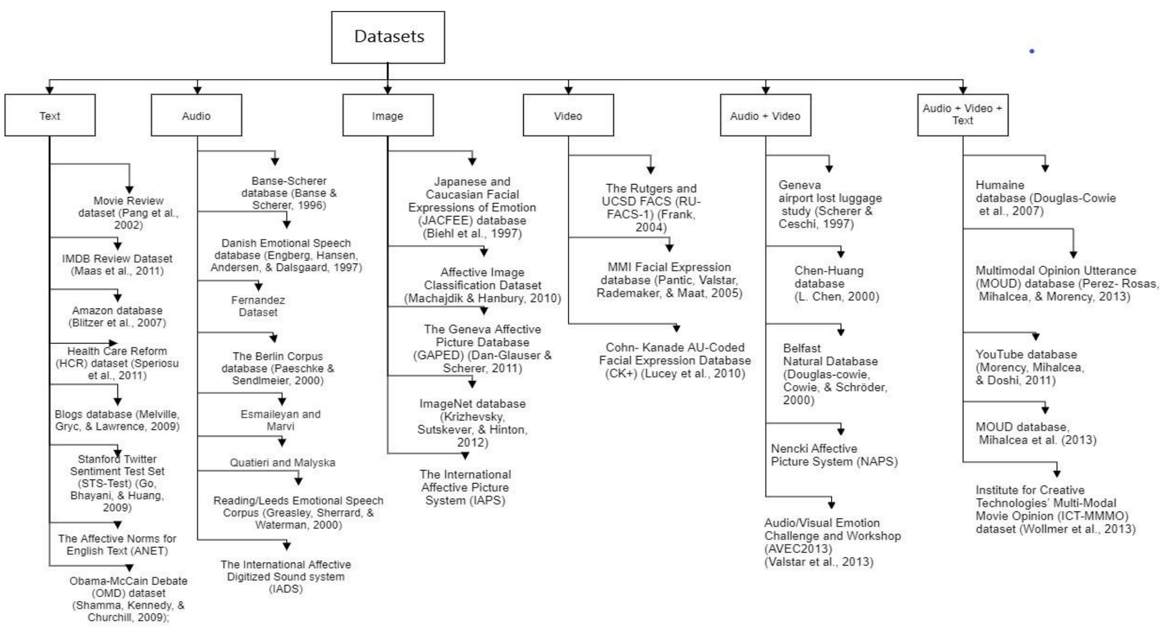
FEASIBILITY STUDY

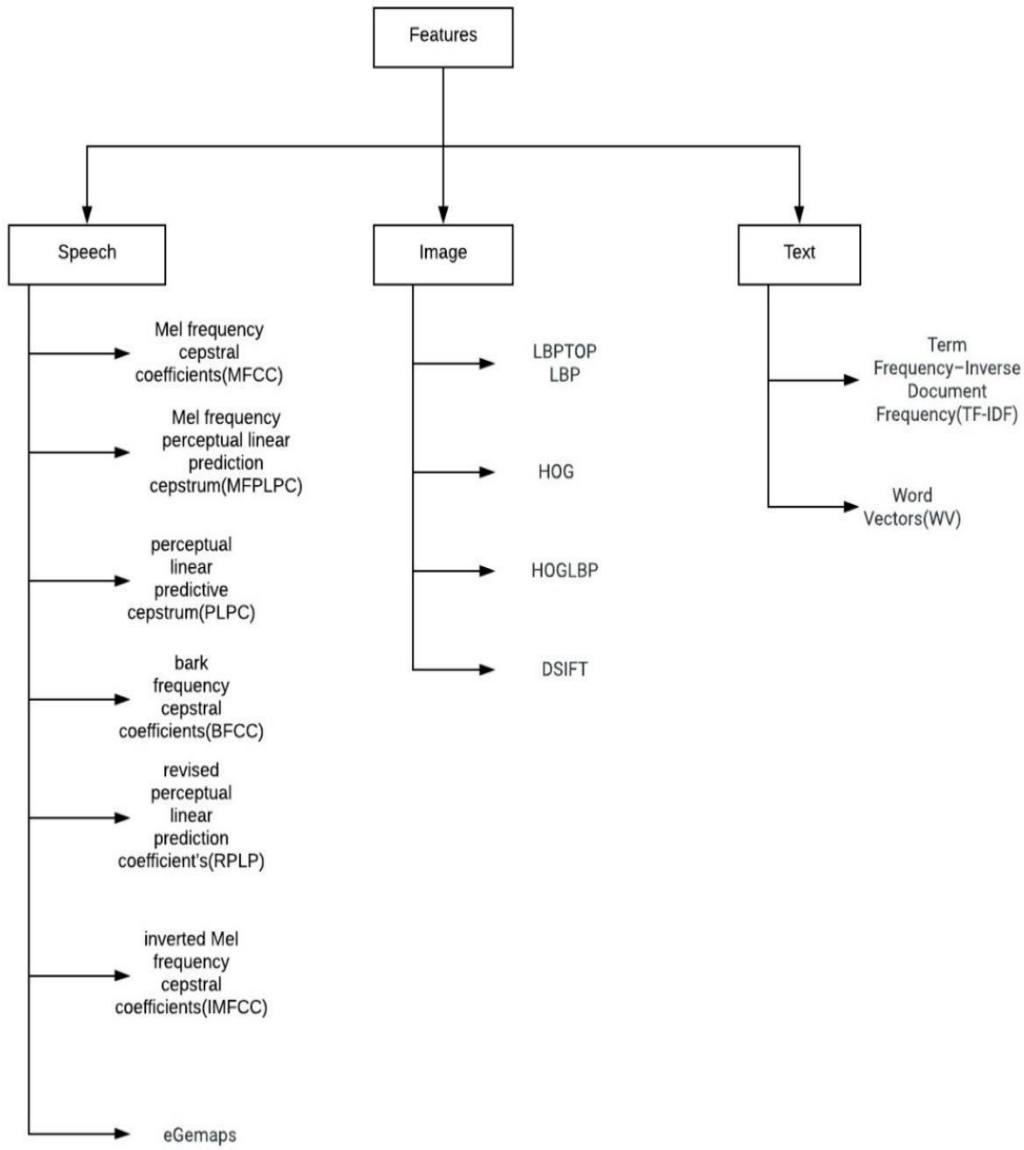
3.1 Description:

This project aims to recognize emotion with different forms of input like speech and facial expressions. By introducing this system in various applications we can find out the emotional status of the person using those applications which will be helpful in many ways.

3.2 Possible Solutions:

Few of them worked on individual inputs and few of them on multiple modes of inputs using different algorithms like RNN (Recurrent neural network), CNN (Convolutional neural network), HMM (Hidden Markov Models), SVMs (Support Vector Machines) etc. They used different features like MFCC, spectral etc. They even used different datasets like Fermandez, Berlin for audio and Fer2013, MMI Facial Expression database etc. for video.





3.3 Most Feasible Solution:

After reviewing all the possible solutions and the efficiencies achieved from different solutions, we came up with a solution of using Ravdess dataset for speech using time distributed CNN which extracts MFCC, Mel-Scale, Spectrogram features. For video Fer2013 dataset using CNN model with synthesized features like “HOG Features, Hybrid Features, Inception, Xception, DeXpression”.

3.4 Conclusion:

So by using the above solution, we will develop a model with better accuracy than the previous models. After development, this model can be integrated in different applications where emotion is recognition is required.

CHAPTER 4
SYSTEM ANALYSIS

4.1. System Requirements

4.1.1. Software Requirements

4.1.1.1. Data sets

Either for synthesis or recognition of emotion by speech or face, we need emotional database for both speech and face. A single actor is not sufficient for recognizing emotions, we need multiple actors to identity emotions and also various styles of expressing the emotions.

While considering database, these are the other factors to be considered. They are:

- size of database
- Speaker/Actor
- gender
- Language (in case of speech system)

A) Speech Emotion Recognition:

In case of Speech Emotion Recognition there are three different methods to collect the data. They are:

- Simulated Databases
- Elicited Databases
- Naturalistic Databases

"Simulated databases are one of major resource of database where actors are asked to portray the given emotion".

“Elicited database are recorded by involving people in particular situation for extracting particular emotion”.

“Naturalistic databases are recorded from our daily life activities”.

There are various data sets for speech emotion recognition. They are: “Ravdess Emotional Speech data set, Emo-DB or Berlin dataset, IEMOCAP” etc., In this project we used “Ravdess data set for Speech Emotion Recognition”.

RAVDESS Emotional Speech Audio:

“Ryerson **A**udio-**V**isual **D**atabase of **E**motional **S**peech and **S**ong” consists of 1440 audio files in .wav form. It contains 24 professional actors i.e., 12 male and 12 female actors. So, “ 60 trials per each actor x 24 actors constitutes 1440 audio files”. We can derive seven emotions like happy, sad, fearful, disgust,calm, angry and surprise expressions from this Ravdess data set. Among 1440 files, each file has its unique name which represents various identifiers of the audio file. They are: “Modality, Vocal Channel, Emotion, Emotional Intensity, Statement, Repetition, Actor”.

B) Face Emotion Recognition:

For Face Emotion Recognition there are many ways to collect the face data. They are:

- Take photographs manually
- Using a program to generate the pictures automatically
- Searching on the internet to collect similar kind of images
- Various data augmentation techniques like flipping the image, rotating the image, cropping the image etc.,

There are many data sets for “emotion recognition using facial expressions”. They are:

“Fer-2013, IEMOCAP, FERET dataset” etc.,

FER- 2013 (“Facial Expression Recognition”) :

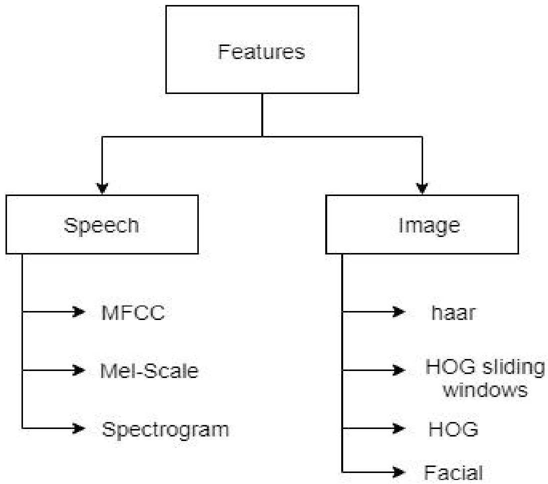
FER data set contains gray scale images of face data with pixel value of 48x48 pixels. There are two .csv files (train and test) in this data set. The train.csv file consists of two columns, “emotion and pixels”. The test.csv file consists of only only one column i.e., “pixles column”. The face data used for training comprises of “28,709 examples”. The face data for testing contains around “3,589 examples”. We can derive seven emotions like happy, sad, fearful, disgust,calm, angry and surprise expressions from this dataset.

Table 4.1 : Datasets

Dataset	Audio	Video	Text
IEMOCAP	✓	✓	✓
CMU-MOSI	✓	✓	✓
AFEW	✓	✓	✓
Fernandez	✓	×	×
RECOLA	✓	✓	×
eNterface	✓	✓	×
Berlin	✓	×	×
Ravdess	✓	×	×
Fer2013	×	✓	×

4.1.1.2. Features:

Feature Extraction plays a vital role in emotion recognition models. In this project we extracted various features for both “Speech Emotion Recognition” and “Emotion Recognition”. For SER, the “Time-distributed CNN” extracts the features like “MFCC, Mel-Scale, Spectrogram” etc., For FER, the CNN synthesized features like “HOG Features, Hybrid Features, Inception, Xception, DeXpression” etc.,



A) Mel Frequency Cepstral Coefficients (MFCC) :

The initial phase in a synthesizing features of SER is to recognize the segments of sound signal that are useful distinguishing the language specialist content. The second step is to remove the background noise and cut the .wav file to particular milli seconds.

“Mel Frequency Cepstral Coefficient” (MFCC) are an element generally utilized in programmed discourse and speaker acknowledgment. They were presented by Davis and Mermelstein in the 1980's, and have been best in class from that point onwards. Preceding the presentation of MFCCS “Linear Expectation Coefficients” (LPCs) and Linear Prediction Cepstral Coefficients (LPCC) and were the fundamental element type for “Automatic Speech Emotion Recognition (ASR)”.

Steps:

- Frame the signal into short frames.
- For each frame compute the periodogram estimate of the power spectrum.
- Apply the Mel filterbank to the power or force spectra, entirety the vitality in each channel.
- Take the logarithm of all filterbank energies.
- Take the DCT of the log filterbank energies.
- Keep DCT coefficients 2-13, dispose of the rest.

B) Mel Scale:

The Mel scale relates apparent recurrence, or pitch of an unadulterated tone to its real estimated recurrence. People are greatly improved at observing little changes in pitch low frequencies than they are at high frequencies. Consolidating this scale makes our features coordinate intimately with human's ear.

The formula for changing over from recurrence to Mel scale is:

M(f) = 1125 In(1 + f/700)   (1)

To go from Mels back to frequency:

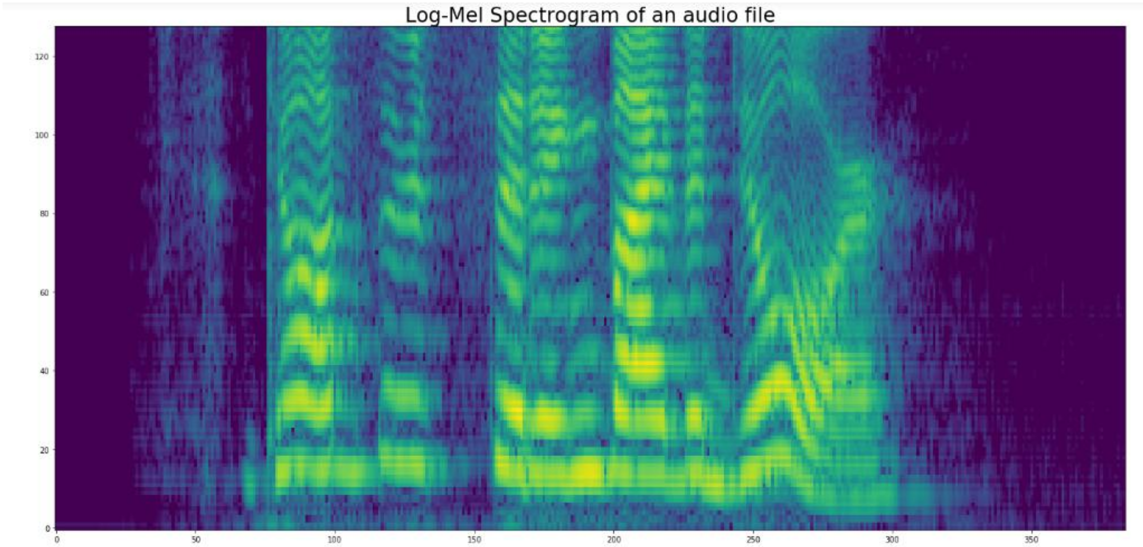
M-1(m) = 700(exp(m/1126) - 1)   (2)

C) **Spectrogram:**

A spectrogram of the discourse signal is a 2D portrayal of the frequencies as for time, that have more data than content translation words for perceiving the feelings of a speaker.

A typical arrangement is a chart with two geometric measurements: one dimension(x-axis) speaks to time, and the other dimension(y-axis) speaks to recurrence; a third measurement(z-axis) demonstrating the abundancy of a specific recurrence at a specific time is spoken to by the power or shade of each point in the picture.

Spectrograms are utilized broadly in the fields of music, sonar, radar, and audio processing, seismology, and others.



D) **Haar features:**

Therefore, a common Haar feature for face detection is a set of two adjacent rectangles that lie above the eye and the cheek region. The position of these rectangles is defined relative to a detection window that acts like a bounding box to the target object (the face in this case).

E) HOG:

Histogram of Oriented Gradients Feature changes over a picture of size width x tallness x 3 (channels) to an element vector/cluster of length n. On account of the HOG descriptor, the information picture is of size 64 x 128 x 3 and the yield include vector is of length 3780.

F) Facial:

Facial expressions are a universal language of emotion, instantly conveying happiness, sadness, anger, fear, and much more. Reading these expressions is essential to compassion and empathy. Some emotions appear more than once.

Other Features:

A smile on human face shows their bliss and it communicates eye with a bended shape. The sad articulation is the inclination of detachment which is ordinarily communicated as rising slanted eyebrows and frown.

The anger on human face is identified with terrible and disturbing conditions. The outflow of annoyance is communicated with pressed eyebrows, thin and extended eyelids. The disgust expressions are communicated with pull down eyebrows and wrinkled nose. The suprise or stun articulation is communicated when some unpredicted occurs. This is communicated with eye-extending and mouth expanding and this expression is an effortlessly recognized one. The statement of fearfulness is connected with shock expression which is communicated as developing slanted eyebrows.

4.1.1.3. Deep Neural Network Models:

In general, there are two methodologies for "Multimodal Emotion Recognition Systems". In the first method, the extracted features are utilized to teach six AI classifiers namely "Support Vector Machines,Random Forest Classifier, Gradient Boosting Classifier, Logistic Regression" etc., while the subsequent methodology depends on deep learning.Emotion Recognition through audio signals using Machine learning classifiers has its own restrictions like, Language, accent dependent. Deep networks may conquer these restrictions. Deep neural networks collect the useful features from the Multimodal data by itself instead of passing the features explicitly to the model.

In the present research findings on Multimodal Emotion Recognition, several works are implemented using various deep neural networks like Recurrent Neural Networks (RNN), Multi-layer perceptron, Long-Short Term Memory Classifier, Convolutional Neural Networks, Time-distributed Neural Networks etc., However, there are few works identified to Hierarchical fusion by combination of one or more modalities like speech+video,text+audio, text+video, audio+video+text etc., In this project, we implemented Time-distributed Neural Networks for Speech Emotion Recognition and Convolutional Neural Networks for Face Emotion Recognition.

For Audio Emotion Recognition, the principle thought of a Time Dispersed Convolutional Neural System is to apply a moving window (fixed size and time-step) up and down the log-mel-spectrogram. Each one of these windows will be the section of a convolutional neural system, created by Four Local Feature Learning Blocks (LFLBs) and the yield of each of these convolutional systems will be taken care of into a repetitive neural system formed by 2 cells of "LSTM" to learn long-term contextual dependencies.

At last, a completely associated layer with softmax activation is utilized to predict the feeling recognized in the voice.

For Facial Emotion Recognition, the aim of using Convolutional Neural Networks is to diminish the pictures into a structure which is simpler to process, without losing features which are basic for getting a decent prediction. We implemented XCeption model to reduce overfitting as much as possible to build a robust model. The XCeption architecture depends on DepthWise Separable convolutions that permit to prepare many less parameters, and in this manner decrease training time on Colab's GPUs to under an hour and a half.

Table 4.2: Models

Models	**Datasets**	**Data**	**Accuracy**
RNN	IEMOCAP	Audio, Video	71.8%
CNN	CMU-MOSI	Audio, Video	49.17%
CNN+LSTM	AFEW	Audio, Video	60.34%
RNN	RECOLA	Audio, Video	76%
CNN+LSTM	eNterface	Audio, Video	90.85%
CNN+RNN	Berlin	Audio	88.9%
RNN	Ravdess	Audio	89.02%
CNN	Fer2013	Video	82.19%

4.1.1.4. Libraries:

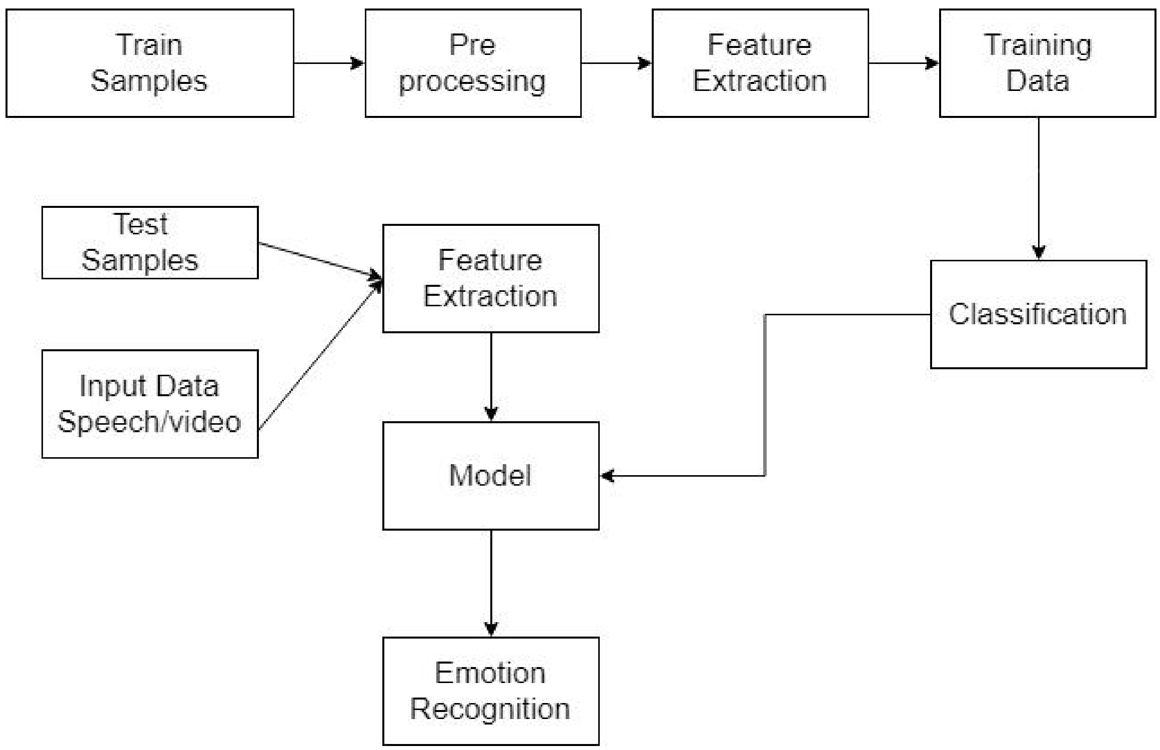
In this project we used several python and machine learning libraries like Librosa, Numpy, Pandas, Soundfile, Pyaudio, Scikit-learn, Tensorflow, Keras, Pytorch , matplotlib etc., for implementing deep neural networks.

4.1.2. Hardware Requirements:

- Processor – i5

- Hard Disk – 5 GB
- Memory – 8 GB RAM
- Nvidia (GPU)

4.1.3. System Architecture:



The proposed human emotion recognition system is of five components:

1. input data

2. pre-processing techniques

3. feature extraction and selection

4. classification

5. emotion recognition

We divide the dataset in the ration of 80:20 (80=training set & 20=testing set). On training dataset pre-processing techniques are applied. From the pre-processed speech signal, we extract the useful features required for emotion recognition.

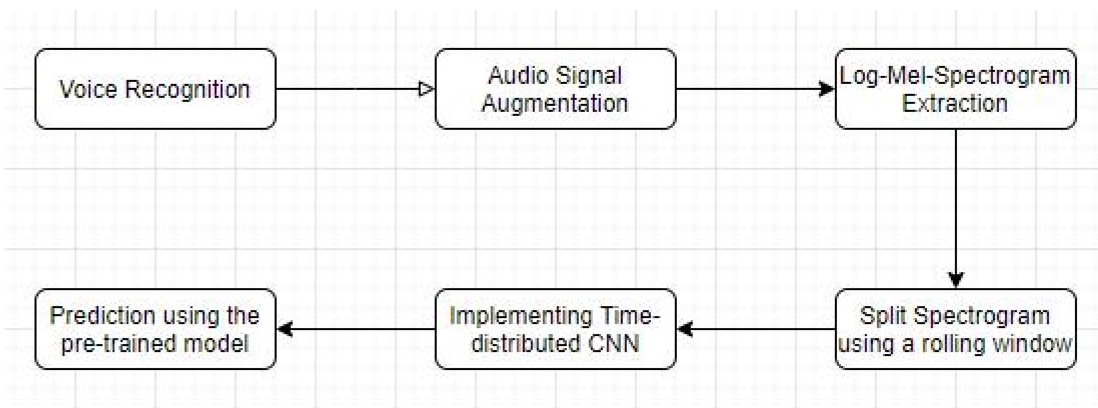
In the next step, Deep Neural Network algorithms like Time-distributed CNN and CNN are implemented on the data. Hence the Emotion Recognition Model is created in .hdf5 or .pickle format.

The input signal is recorded and we synthesize the features which are useful for emotion recognition.The recorded data is given as input for the emotion recognition model. In the final stage, one of the 7 emotions is identified and displayed on the webpage.

4.1.4. Methodology:

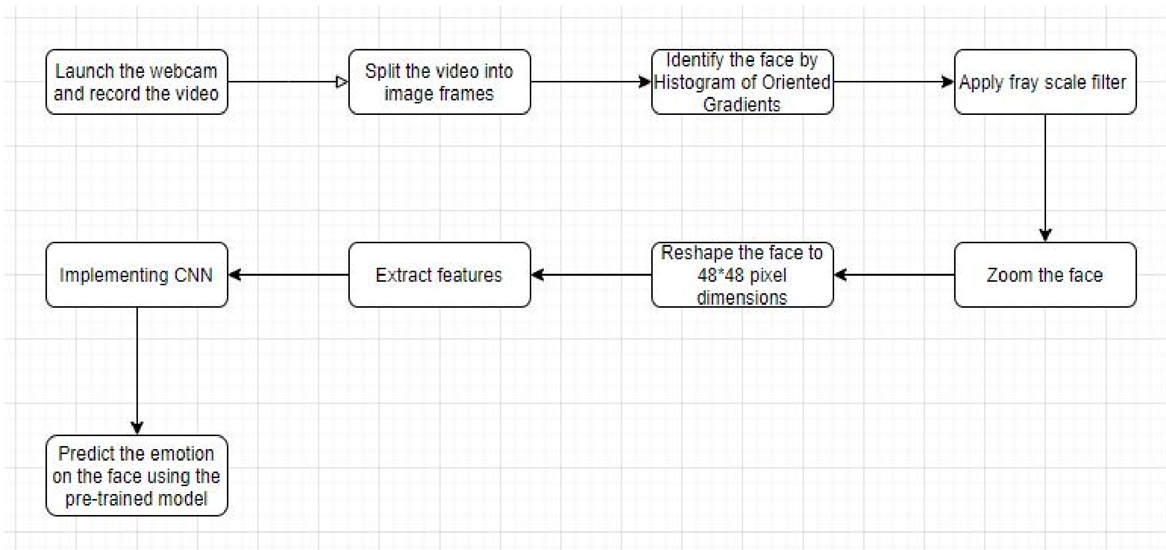
4.1.4.1. Speech Emotion Recognition:

The Speech Emotion Recognition methodology pipeline is built in the following way:



4.1.4.2. Face Emotion Recognition:

The Face Emotion Recognition methodology Pipeline is built in the following way:



CHAPTER 5
SOFTWARE DESIGN

5.1 UML Diagrams

Unified Modelling Language is a tool that assists a designer to present his thoughts about the task to his customer and his designer. Modelling theaters a vital role in scheming a software. A sick designed model can lead to a poorly settled software.

A UML system has using five diverse views that support in labelling systems from diverse outlooks. Each view has a set of diagrams that and modules that represent the real time objects.

a. User Model View:

i. It models the operator behavior in a organization context.

ii. All the diagrams are drawn keeping in mind the user's response and reaction towards a system.

b. Structural Model View

i. This vision contains of class and object diagram which is used to model the static structures.

ii. It uses objects, operations, attributes and relationships.

c. Behavioral Model View

i. It largely contains the sequence diagram, collaboration diagram, state chart diagram and activity diagram. They mostly symbolize flow of movements among different objects tangled in the system.

ii. They are used to envision numerous lively aspects of the system architecture.

d. Implementation Model View

a. This outlook consists of component diagrams and deployment diagrams. This view models the static software modules for an organization.

b. This usually contains the data files, the executables, documentation and source code.

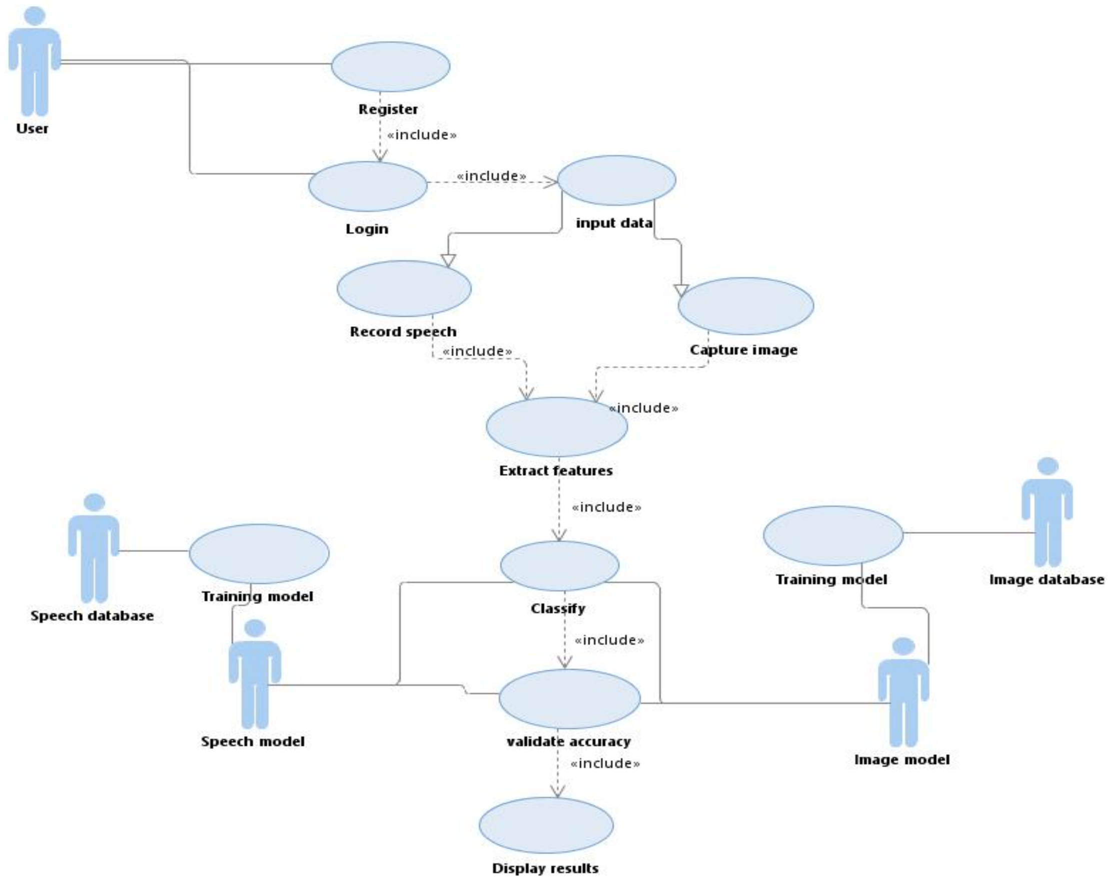
c. These are the materially useable components of the system. They are modelled using component diagrams.

5.1.1. Use Case Diagram

The simple representation for the collaboration of the user with the system is symbolized by the use case diagram. It includes the connection between the user and many use cases with the actors being involved. There are diverse kind of relations that are involved among the use cases and the actors. They include:

a. Association relationship
b. Generalization
c. Dependency
d. Realizations
e. Transitions

The following represents the use case diagram of the proposed system:



5.1.2. Class Diagram

They are stationary representation of an solicitation. Lone the class diagrams have the ability to be straight mapped with the OOP Languages since in OOPs the whole thing is model in the usage of classes and objects. As of this intention these diagrams are used extensively at the time of building. This is one of the most commonly used UML diagram in the exclusive community. A class diagram plays an crucial role in forward and reverse engineering.

a. It deeds as a base for the component and deployment diagrams.

b. It mostly defines and expresses the simple tasks of a system's application.

c. It gears the scrutiny and design view for a static application.

In a class diagram, every object is modelled as a class. Each class comprises of section or compartments.

1. Class name
2. Attributes of a class or operations
3. Methods or functions
4. Documentation (optional section)

The subsequent points have to to be summon up while drawing a class diagram:

a. The labels of the class diagram need to be significant to depict the feature of the framework.
b. Every component and their relations must be illustrious gaining of time.
c. Every class has a accountability (attributes and methods) that need to be recognized openly.
d. Total properties for all classes need to be minimum. Subsequently useless properties will create the diagram intricate.
e. At all opinion required to show some part of the diagram use notes Since to the completion of the diagram it should be reasonable to the designer/coder.
f. Afore concluding the previous version, the diagram must be drawn on normal paper and review whatever figure conditions as would be sensible to make it amends.

1. Scopes:

The UML diagrams have two types of possibilities for class members:

i. instance members scope and
ii. classifier members scope

2. Classifier members are "static" members of a class in various programming languages. The scope is the class itself.

i. Static attributes are mutual to all additional objects that raise the class.

ii. Static methods are not instantiated.

3. Instance members are the members that are local to an object.

i. The key purpose of instance members is to allow the objects to store their states.

ii. Statements exterior the methods are generally famous as instance members.

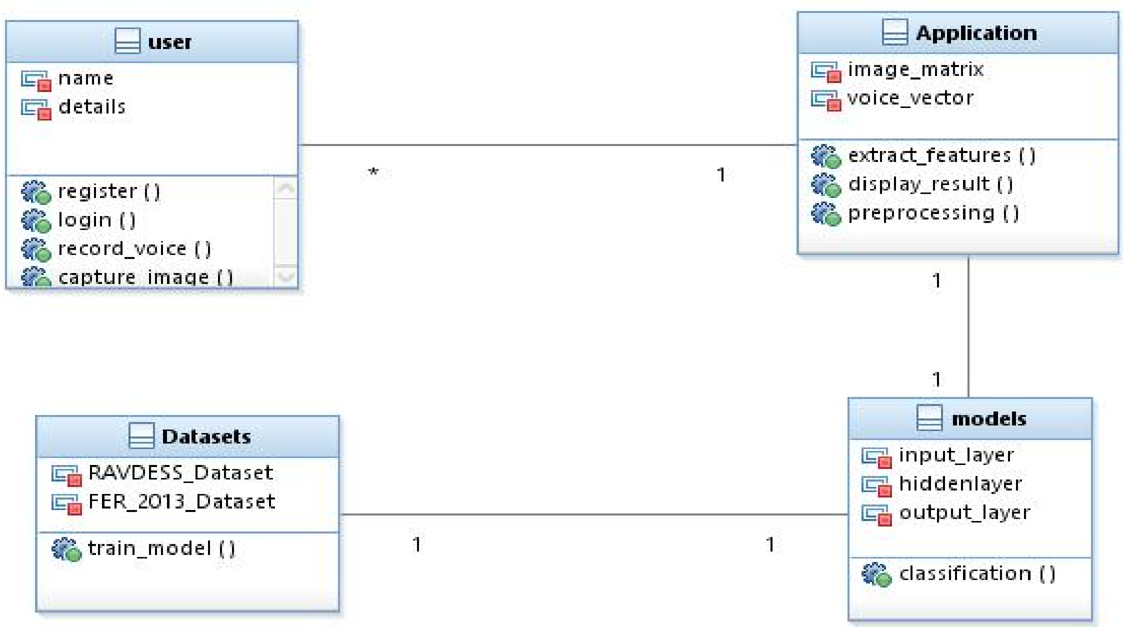


Fig 5.2: Class Diagram for Developed Model

5.1.3. Sequence Diagram

The Sequence Diagram portrays the time series amid many objects in an application. It shows the series of communications with which objects connect with each other so that they carry out the requisite functionality.

It contains of the salvations which are generally parallel vertical lines. It comprises of horizontal arrows which specifies the route of the communications that are swapped in a right order which makes the user cool to understand.

The lifeline for a given object represents a role. The synchronous calls are represented with the help of a solid arrow head whereas the asynchronous messages are represented with the help of open arrow heads.

All objects are represented according to their time ordering. Timing of messages plays a major role in sequence diagrams. An object is killed immediately after its use in sequence diagrams.

I). Common Properties:

An arrangement graph is much the same as unique sort of diagram and offers some indistinguishable properties from other diagrams. In any case, it varies from every single other diagram in its content.

II). Contents

Objects are normally named or unknown instances of class, however may likewise speak to occurrences of different things, for example components, collaboration and nodes. Graphically, object is represented as a rectangle by underlying its name.

III). Links

A link is a semantic association among objects i.e., an object of an affiliation is called as a connection. It is represented as a line.

IV). Messages

A message is a determination of a correspondence between objects that passes on the data with the desire that the action will follow.

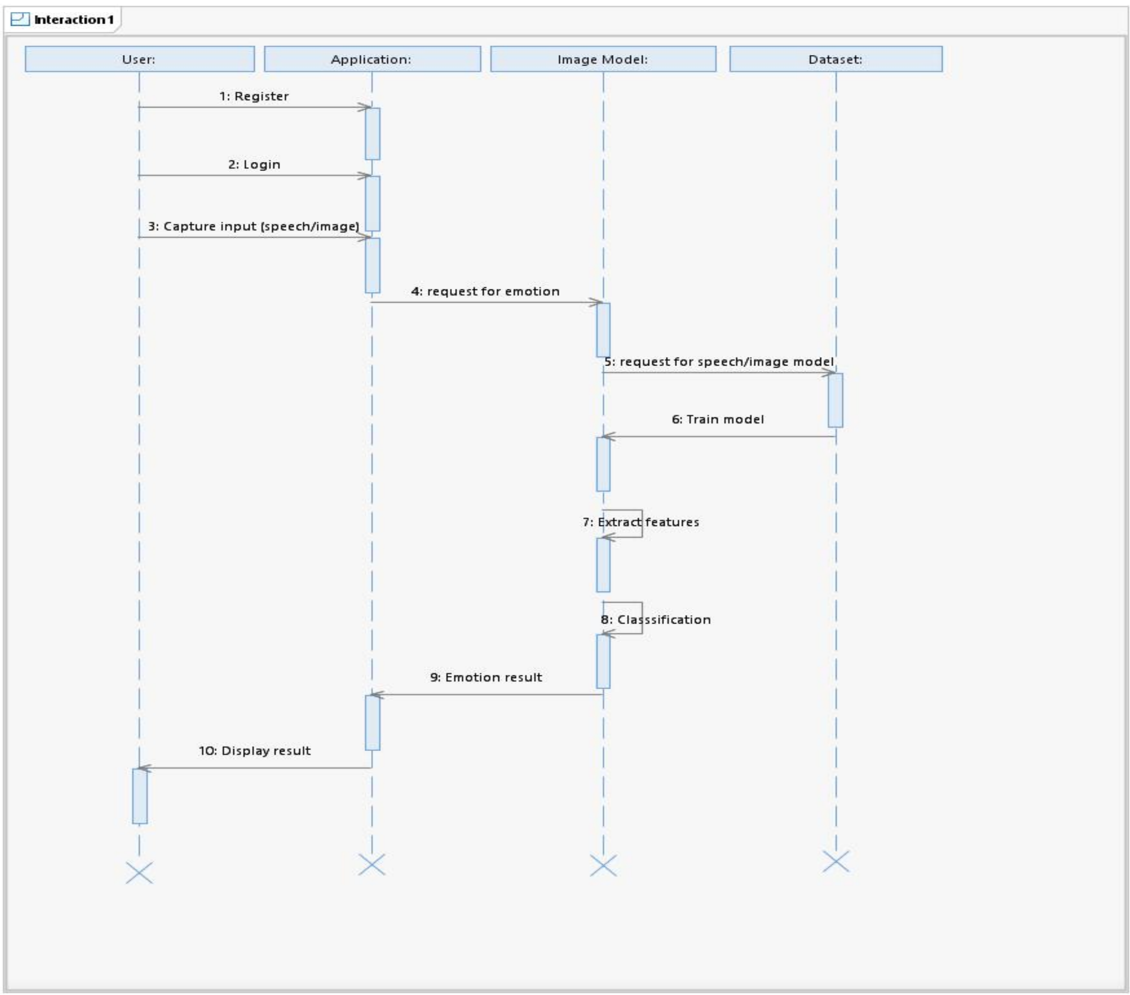


Fig 5.3 Sequence Diagram for Developed Model

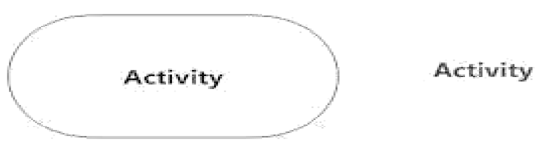
5.1.4. Activity Diagram

The flow from one activity to another activity can be represented in the form of a flow chart which is usually an activity diagram. It forms a backbone for the UML diagrams. It depicts the dynamic aspects for all the objects within the system.

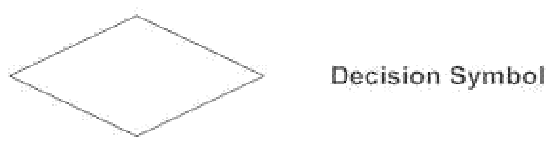
The control flow from one object to another object is drawn which shows the basic operations that are to be performed.

Activity diagrams are constructed using the following:

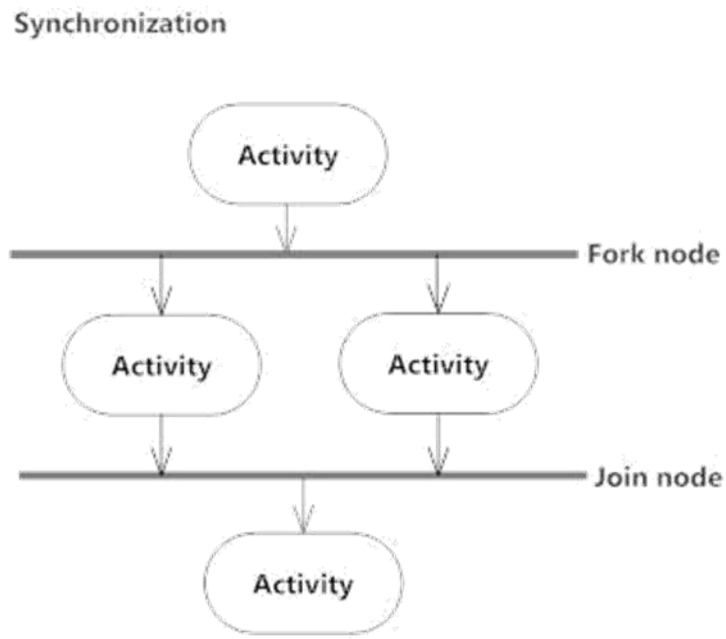
1. Actions are represented using rounded rectangles;



2. decisions are represented using diamonds;



3. concurrent activities bars are represented using the start (split) or end (join) ;



4. Time event is represented as

5. final state is represented using encircled black circle.

The basic purpose of an activity diagram is same as that of other UML diagrams. The dynamic behavior of the system is viewed by the activity diagram. They are used to construct a system using the backward and forward engineering mechanisms.

The purpose of an activity diagram is as follows:

1) For drawing the flow (i.e. activity) in a system.
2) For showing the flow of sequence from one activity to another activity.
    1) For showing the concurrent and parallel flow of actions in the system.
    The elements that are used in an activity diagram are as follows:

        i. Association Relationship
        ii. Activities
        iii. Conditions and Constraints.

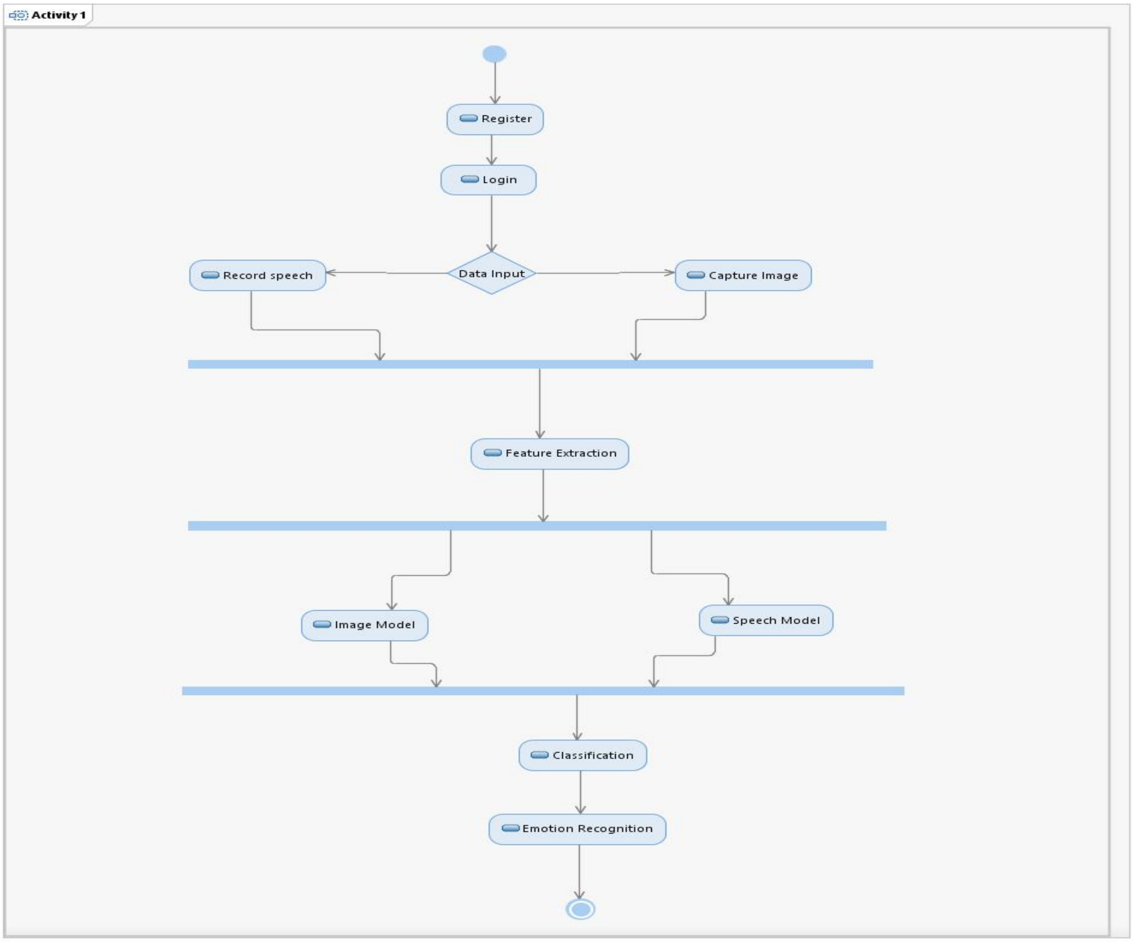


Fig 5.4 Activity Diagram

CHAPTER 6
IMPLEMENTATION

6.1. Source Code and Outputs

6.1.1. Speech Emotion Recognition

6.1.1.1. Signal Processing

The Signal Processing includes:

- Signal Discretization
- Audio Data Augmentation
- Log-mel-spectrogram extraction
- Time distributes framing
- Build train and test data set

```
# General imports

import os

from glob import glob

import pickle

import itertools

import numpy as np
```

```
from scipy.stats import zscore
from sklearn.model_selection import train_test_split


# Graph imports
import matplotlib.pyplot as plt
from PIL import Image


# Audio imports
import librosa
import IPython
from IPython.display import Audio


#Set Labels
# RAVDESS Database
label_dict_ravdess = {'02': 'NEU', '03':'HAP', '04':'SAD', '05':'ANG', '06':'FEA', '07':'DIS',
'08':'SUR'}


# Set audio files labels
def set_label_ravdess(audio_file, gender_differentiation):
    label = label_dict_ravdess.get(audio_file[6:-16])
    if gender_differentiation == True:
        if int(audio_file[18:-4])%2 == 0: # Female
            label = 'f_' + label
```

```
    if int(audio_file[18:-4])%2 == 1: # Male
        label = 'm_' + label
return label


#Import Audio Files
# Start feature extraction
print("Import Data: START")


# Audio file path and names
#file_path = '../Datas/Ravdess/'
#file_names = os.listdir(file_path)
#print(file_path)
#print(file_names)
audio_filepath=''
audio_fileslist=[]
#for i in range(1,25):
    #audio_filepath='../Datas/Ravdess/Actor_'+str(i)+'/'
    #print(audio_filepath)
    #audio_fileslist=os.listdir(audio_filepath)
    #for i in audio_fileslist:
        #print(i)
#print(audio_fileslist)
# Initialize features and labels list
```

```
signal = []
labels = []
# Sample rate (16.0 kHz)
sample_rate = 16000
# Max pad lenghth (3.0 sec)
max_pad_len = 49100
# Compute spectogram for all audio file
for i in range(1,25):
  audio_filepath='../Datas/Ravdess/Actor_'+str(i)+'/'
  audio_fileslist=os.listdir(audio_filepath)
  for audio_index, audio_file in enumerate(audio_fileslist):
  #print(audio_file)
    #print(audio_index)
    if audio_file[6:-16] in list(label_dict_ravdess.keys()):
    #print(audio_file[6:-16])
    # Read audio file
      y, sr = librosa.core.load(audio_filepath + audio_file, sr=sample_rate, offset=0.5)
    # Z-normalization
      y = zscore(y)
    # Padding or truncated signal
      if len(y) < max_pad_len:
        y_padded = np.zeros(max_pad_len)
        y_padded[:len(y)] = y
```

```
            y = y_padded
          elif len(y) > max_pad_len:
            y = np.asarray(y[:max_pad_len])

        # Add to signal list
          signal.append(y)
        # Set label
          labels.append(set_label_ravdess(audio_file,False))
          #print(audio_index)
        # Print running...
          if (audio_index % 100 == 0):
            print("Import Data: RUNNING ... {} files".format(audio_index))
# Cast labels to array
labels = np.asarray(labels).ravel()
#print(labels)
#print(signal)
# Stop feature extraction
print("Import Data: END \n")
print("Number of audio files imported: {}".format(labels.shape[0]))

# Select one random audio file
x=[]
print(len(labels))
```

```
random_idx = np.random.randint(len(labels))
print(random_idx)
random_label = labels[random_idx]
print(random_label)
random_signal = signal[random_idx]
print(random_signal)
for i in range(1,25):
    audio_filepath='../Datas/Ravdess/Actor_'+str(i)+'/'
    audio_fileslist=os.listdir(audio_filepath)
    #print(audio_fileslist)
    x=x+audio_fileslist
    #print(x)
random_filename = x[random_idx]
print(random_filename)


# Plot signal wave
plt.figure(figsize=(10,5))
plt.plot(np.arange(len(random_signal))/float(sample_rate), random_signal)
plt.xlim((np.arange(len(random_signal))/float(sample_rate))[0], (np.arange(len(rando
m_signal))/float(sample_rate))[-1])
plt.xlabel('Time (s)', fontsize=16)
plt.ylabel('Amplitude (dB)', fontsize=16)
plt.title("Signal wave of file '{}' with label {}".format(random_filename, random_label)
```

```
, fontsize=18)
    plt.show()


    # Play audio file
    print("Audio file '{}':".format(random_filename))
    Audio(random_signal, rate=sample_rate)


    #Audio Data Augmentation
    # Number of augmented data
    nb_augmented = 2


    # Function to add noise to a signals with a desired Signal Noise ratio (SNR)
    def noisy_signal(signal, snr_low=15, snr_high=30, nb_augmented=2):


        # Signal length
        signal_len = len(signal)
        # Generate White noise
        noise = np.random.normal(size=(nb_augmented, signal_len))
        # Compute signal and noise power
        s_power = np.sum((signal / (2.0 ** 15)) ** 2) / signal_len
        n_power = np.sum((noise / (2.0 ** 15)) ** 2, axis=1) / signal_len
        # Random SNR: Uniform [15, 30]
        snr = np.random.randint(snr_low, snr_high)
```

```
        # Compute K coeff for each noise
        K = np.sqrt((s_power / n_power) * 10 ** (- snr / 10))
        K = np.ones((signal_len, nb_augmented)) * K
        # Generate noisy signal
        return signal + K.T * noise
    # Generate noisy signals from signal list
    print("Data Augmentation: START")
    augmented_signal = list(map(noisy_signal, signal))
    print("Data Augmentation: END!")


    # Plot signal wave
    plt.figure(figsize=(20,5))
    plt.subplot(1,2,1)
    plt.plot(np.arange(len(random_signal))/float(sample_rate), random_signal)
    plt.xlim((np.arange(len(random_signal))/float(sample_rate))[0], (np.arange(len(rando
m_signal))/float(sample_rate))[-1])
    plt.xlabel('Time (s)', fontsize=16)
    plt.ylabel('Amplitude (dB)', fontsize=16)
    plt.title("Signal wave of file '{}' ".format(random_filename), fontsize=18)


    # Plot signal wave with noise
    plt.subplot(1,2,2)
    plt.plot(np.arange(len(random_signal))/float(sample_rate), augmented_signal[random_
```

```
idx][0])
    plt.xlim((np.arange(len(random_signal))/float(sample_rate))[0], (np.arange(len(rando
m_signal))/float(sample_rate))[-1])
    plt.xlabel('Time (s)', fontsize=16)
    plt.ylabel('Amplitude (dB)', fontsize=16)
    plt.title("Signal wave of file '{}' with Noise".format(random_filename), fontsize=18)
    plt.show()
    # Play audio file
    print("Audio file '{}':".format(random_filename))
    IPython.display.display(Audio(random_signal, rate=sample_rate))

    # Play same audio file with noise
    print("Audio file '{}' with noise:".format(random_filename))
    IPython.display.display(Audio(augmented_signal[random_idx][0], rate=sample_rate))

    #Feature Extraction
    def mel_spectrogram(y, sr=16000, n_fft=512, win_length=256, hop_length=128, wind
ow='hamming', n_mels=128, fmax=4000):

        # Compute spectogram
        mel_spect = np.abs(librosa.stft(y, n_fft=n_fft, window=window, win_length=win_le
ngth, hop_length=hop_length)) ** 2
```

```
        # Compute mel spectrogram
        mel_spect = librosa.feature.melspectrogram(S=mel_spect, sr=sr, n_mels=n_mels, f
max=fmax)

        # Compute log-mel spectrogram
        mel_spect = librosa.power_to_db(mel_spect, ref=np.max)
        return mel_spect

    # Start feature extraction
    print("Feature extraction: START")
    # Compute spectogram for all audio file
    mel_spect = np.asarray(list(map(mel_spectrogram, signal)))
    augmented_mel_spect = [np.asarray(list(map(mel_spectrogram, augmented_signal[i])))
 for i in range(len(augmented_signal))]
    # Stop feature extraction
    print("Feature extraction: END!")
    # Plot one random Spectogram
    plt.figure(figsize=(20, 10))
    plt.imshow(mel_spect[np.random.randint(len(mel_spect))], origin='lower', aspect='aut
o', cmap='viridis')
    plt.title('Log-Mel Spectrogram of an audio file', fontsize=26)
    plt.tight_layout()
    plt.show()
```

```
#Train and Test set

# Build Train and test dataset

MEL_SPECT_train, MEL_SPECT_test, AUG_MEL_SPECT_train, AUG_MEL_SPE
CT_test, label_train, label_test = train_test_split(mel_spect, augmented_mel_spect, labels,
 test_size=0.2)


# Build augmented labels and train

aug_label_train = np.asarray(list(itertools.chain.from_iterable([[label] * nb_augmented
 for label in label_train])))

AUG_MEL_SPECT_train = np.asarray(list(itertools.chain.from_iterable(AUG_MEL_
SPECT_train)))


# Concatenate original and augmented

X_train = np.concatenate((MEL_SPECT_train, AUG_MEL_SPECT_train))

y_train = np.concatenate((label_train, aug_label_train))


# Build test set

X_test = MEL_SPECT_test

y_test = label_test


# Delete

del MEL_SPECT_train, AUG_MEL_SPECT_train, label_train, aug_label_train, AUG
_MEL_SPECT_test, MEL_SPECT_test, label_test
```

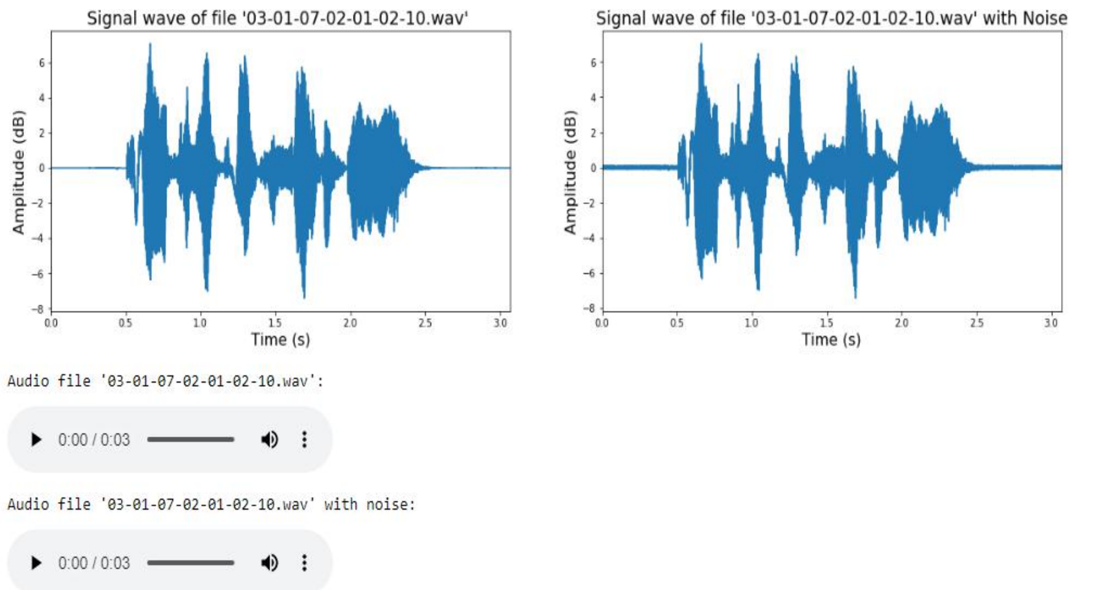
```
del mel_spect, augmented_mel_spect, labels

#Time distributed Framing

# Time distributed parameters

win_ts = 128

hop_ts = 64

# Split spectrogram into frames

def frame(x, win_step=128, win_size=64):

    nb_frames = 1 + int((x.shape[2] - win_size) / win_step)

    frames = np.zeros((x.shape[0], nb_frames, x.shape[1], win_size)).astype(np.float32)

    for t in range(nb_frames):

        frames[:,t,:,:] = np.copy(x[:,:,(t * win_step):(t * win_step + win_size)]).astype(np.
float32)

    return frames


# Frame for TimeDistributed model

X_train = frame(X_train, hop_ts, win_ts)

X_test = frame(X_test, hop_ts, win_ts)


#Saving

# Save Train and test set

pickle.dump(X_train.astype(np.float16), open('../Datas/Pickle/RAVDESS/[RAVDESS]
[MEL_SPECT][X_train].p', 'wb'))

pickle.dump(y_train, open('../Datas/Pickle/RAVDESS/[RAVDESS][MEL_SPECT][y_
```

train].p', 'wb'))

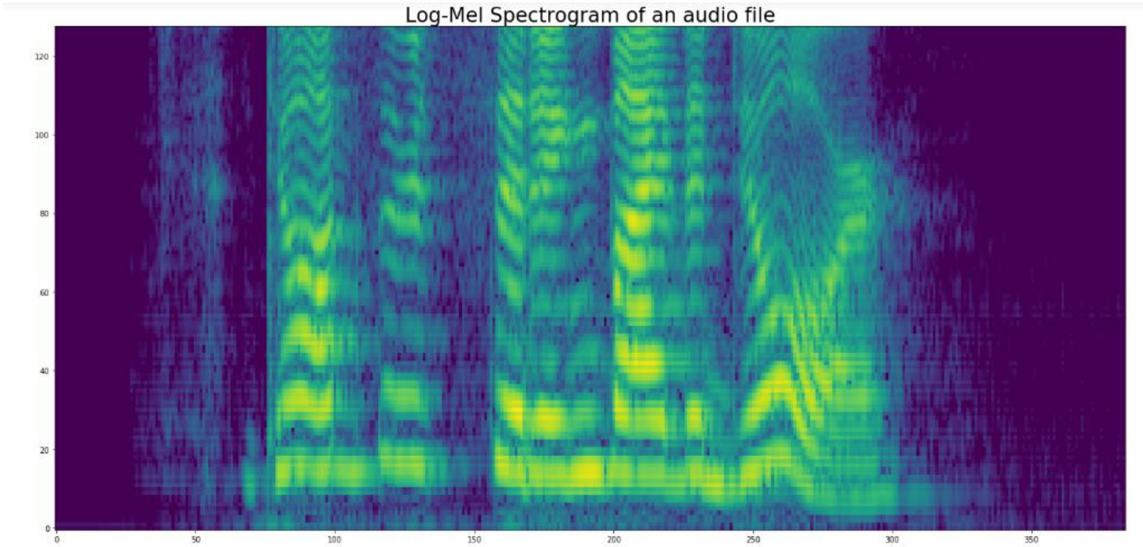
pickle.dump(X_test.astype(np.float16), open('../Datas/Pickle/RAVDESS/[RAVDESS][MEL_SPECT][X_test].p', 'wb'))

pickle.dump(y_test, open('../Datas/Pickle/RAVDESS/[RAVDESS][MEL_SPECT][y_test].p', 'wb'))

Output of Data Augmentation:



Output of Spectrogram:



6.1.1.2. Time Distributed Neural Network

#General Imports

import os

from glob import glob

import pickle

import numpy as np

Plot imports

from IPython.display import Image

import matplotlib.pyplot as plt

Time Distributed ConvNet imports

import tensorflow as tf

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Input, Dense, Dropout, Activation, TimeDistributed, concatenate

from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, Batch

```
Normalization, LeakyReLU, Flatten

from tensorflow.keras.layers import LSTM

from tensorflow.keras.optimizers import Adam, SGD

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROn
Plateau

from tensorflow.keras import backend as K

from keras.utils import np_utils

from keras.utils import plot_model

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.utils import plot_model

#Warning imports

import warnings

warnings.filterwarnings('ignore')

# Connect Colab to google drive

from google.colab import drive

drive.mount('/content/drive')


#Import datas

# RAVDESS mel-Spectrogram

X_train = pickle.load(open('drive/My Drive/SpeechEmotionRecognition/[RAVDESS][M
EL_SPECT][X_train].p', 'rb'))

y_train = pickle.load(open('drive/My Drive/SpeechEmotionRecognition/[RAVDESS][M
EL_SPECT][y_train].p', 'rb'))
```

```
y_test = pickle.load(open('drive/My Drive/SpeechEmotionRecognition/[RAVDESS][ME
L_SPECT][y_test].p', 'rb'))

X_test = pickle.load(open('drive/My Drive/SpeechEmotionRecognition/[RAVDESS][ME
L_SPECT][X_test].p', 'rb'))

#Encode Label

# Encode Label from categorical to numerical

lb = LabelEncoder()

y_train = np_utils.to_categorical(lb.fit_transform(np.ravel(y_train)))

y_test = np_utils.to_categorical(lb.transform(np.ravel(y_test)))

#Reshape train and test set

# Reshape for convolution

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1] , X_train.shape[2], X_train.
shape[3], 1)

X_test = X_test.reshape(X_test.shape[0], X_test.shape[1] , X_test.shape[2], X_test.shape
[3], 1)

#Time-distributed Convolutional Neural Network Model

K.clear_session()

# Define two sets of inputs: MFCC and FBANK

input_y = Input(shape=X_train.shape[1:], name='Input_MELSPECT')


## First LFLB (local feature learning block)

y = TimeDistributed(Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='same'), na
me='Conv_1_MELSPECT')(input_y)

y = TimeDistributed(BatchNormalization(), name='BatchNorm_1_MELSPECT')(y)
```

y = TimeDistributed(Activation('elu'), name='Activ_1_MELSPECT')(y)

y = TimeDistributed(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'), name='MaxPool_1_MELSPECT')(y)

y = TimeDistributed(Dropout(0.2), name='Drop_1_MELSPECT')(y)

Second LFLB (local feature learning block)

y = TimeDistributed(Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='same'), name='Conv_2_MELSPECT')(y)

y = TimeDistributed(BatchNormalization(), name='BatchNorm_2_MELSPECT')(y)

y = TimeDistributed(Activation('elu'), name='Activ_2_MELSPECT')(y)

y = TimeDistributed(MaxPooling2D(pool_size=(4, 4), strides=(4, 4), padding='same'), name='MaxPool_2_MELSPECT')(y)

y = TimeDistributed(Dropout(0.2), name='Drop_2_MELSPECT')(y)

Second LFLB (local feature learning block)

y = TimeDistributed(Conv2D(128, kernel_size=(3, 3), strides=(1, 1), padding='same'), name='Conv_3_MELSPECT')(y)

y = TimeDistributed(BatchNormalization(), name='BatchNorm_3_MELSPECT')(y)

y = TimeDistributed(Activation('elu'), name='Activ_3_MELSPECT')(y)

y = TimeDistributed(MaxPooling2D(pool_size=(4, 4), strides=(4, 4), padding='same'), name='MaxPool_3_MELSPECT')(y)

y = TimeDistributed(Dropout(0.2), name='Drop_3_MELSPECT')(y)

Second LFLB (local feature learning block)

```
y = TimeDistributed(Conv2D(128, kernel_size=(3, 3), strides=(1, 1), padding='same'), na
me='Conv_4_MELSPECT')(y)

y = TimeDistributed(BatchNormalization(), name='BatchNorm_4_MELSPECT')(y)

y = TimeDistributed(Activation('elu'), name='Activ_4_MELSPECT')(y)

y = TimeDistributed(MaxPooling2D(pool_size=(4, 4), strides=(4, 4), padding='same'), na
me='MaxPool_4_MELSPECT')(y)

y = TimeDistributed(Dropout(0.2), name='Drop_4_MELSPECT')(y)

## Flat

y = TimeDistributed(Flatten(), name='Flat_MELSPECT')(y)

# Apply 2 LSTM layer and one FC

y = LSTM(256, return_sequences=False, dropout=0.2, name='LSTM_1')(y)

y = Dense(y_train.shape[1], activation='softmax', name='FC')(y)

# Build final model

model = Model(inputs=input_y, outputs=y)

# Plot model graph

plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')

Image(retina=True, filename='model.png')

# Compile model

model.compile(optimizer=SGD(lr=0.01, decay=1e-
6, momentum=0.8), loss='categorical_crossentropy', metrics=['accuracy'])

# Save best model

best_model_save = ModelCheckpoint('drive/My Drive/SpeechEmotionRecognition/[CN
N-LSTM]Model.hdf5', save_best_only=True, monitor='val_acc', mode='max')

# Early stopping
```

```
early_stopping = EarlyStopping(monitor='val_acc', patience=30, verbose=1, mode='max')

# Fit model

history = model.fit(X_train, y_train, batch_size=64, epochs=100, validation_data=(X_test,
 y_test), callbacks=[early_stopping, best_model_save])

# Loss Curves

plt.figure(figsize=(25, 10))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'],'-g',linewidth=1.0)

plt.plot(history.history['val_loss'],'r',linewidth=1.0)

plt.legend(['Training loss', 'Validation Loss'],fontsize=14)

plt.xlabel('Epochs ',fontsize=16)

plt.ylabel('Loss',fontsize=16)

plt.title('Loss Curves',fontsize=22)


# Accuracy Curves

plt.subplot(1, 2, 2)

plt.plot(history.history['acc'],'-g',linewidth=1.0)

plt.plot(history.history['val_acc'],'r',linewidth=1.0)

plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=14)

plt.xlabel('Epochs ',fontsize=16)

plt.ylabel('Accuracy',fontsize=16)

plt.title('Accuracy Curves',fontsize=22)

plt.show()
```

#Save the model

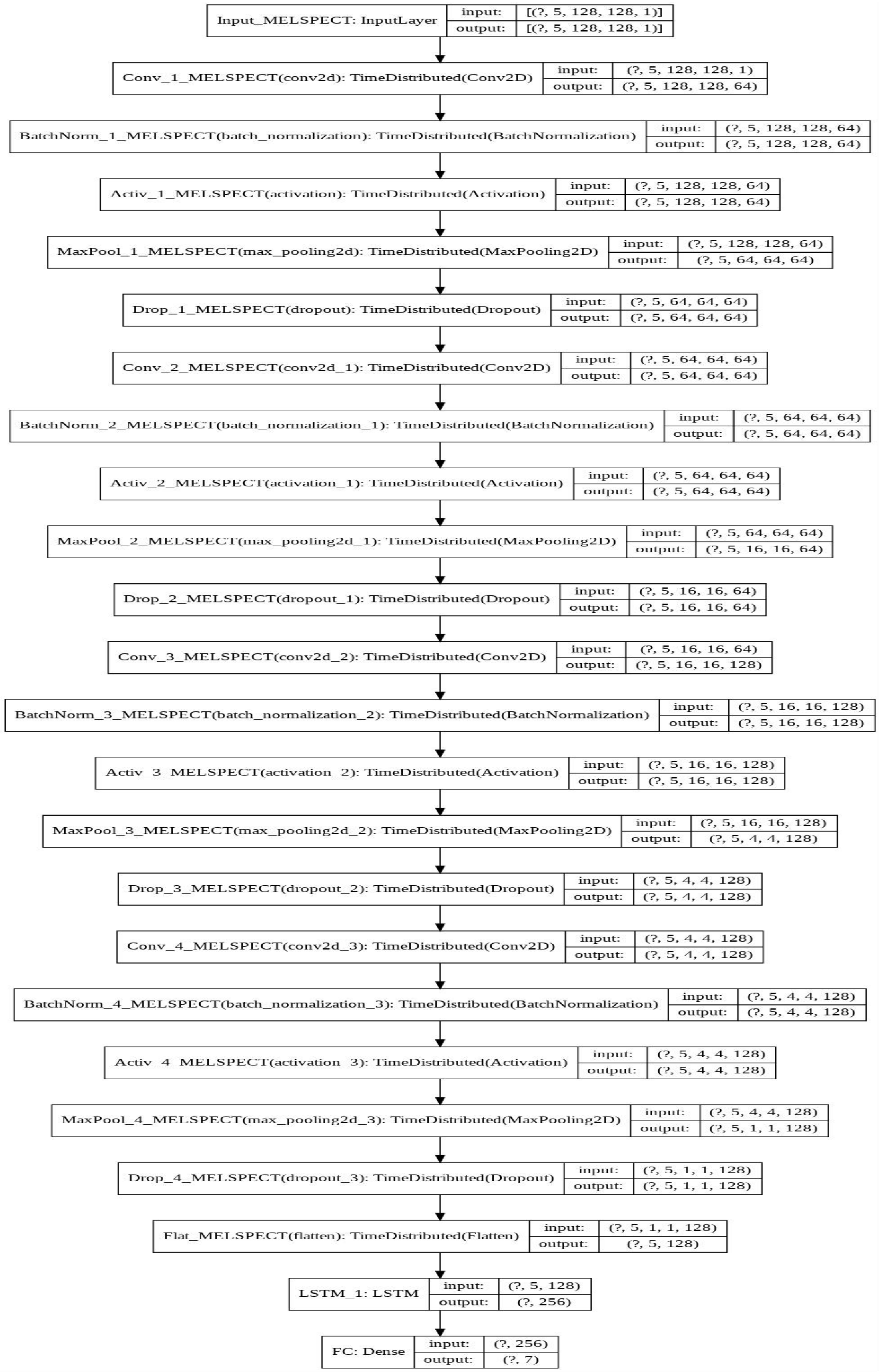
model.save('drive/My Drive/SpeechEmotionRecognition/[CNN-LSTM]M.h5')

model.save_weights('drive/My Drive/SpeechEmotionRecognition/[CNN-LSTM]W.h5')

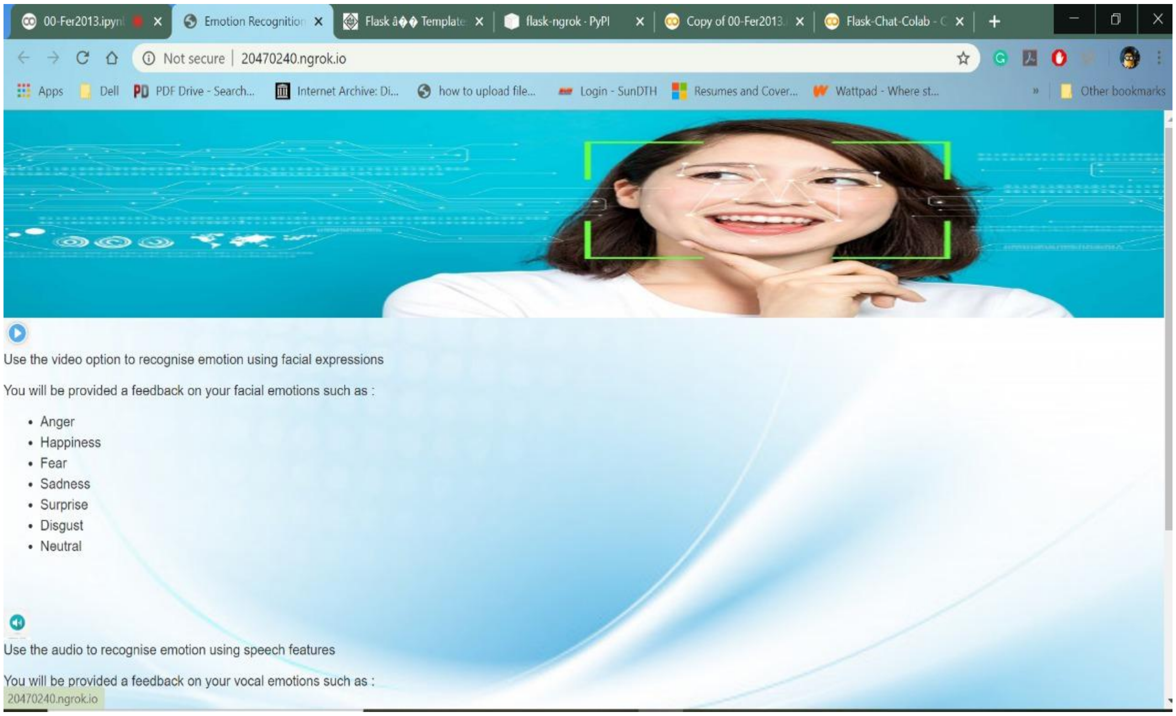
Output of Accuracy:

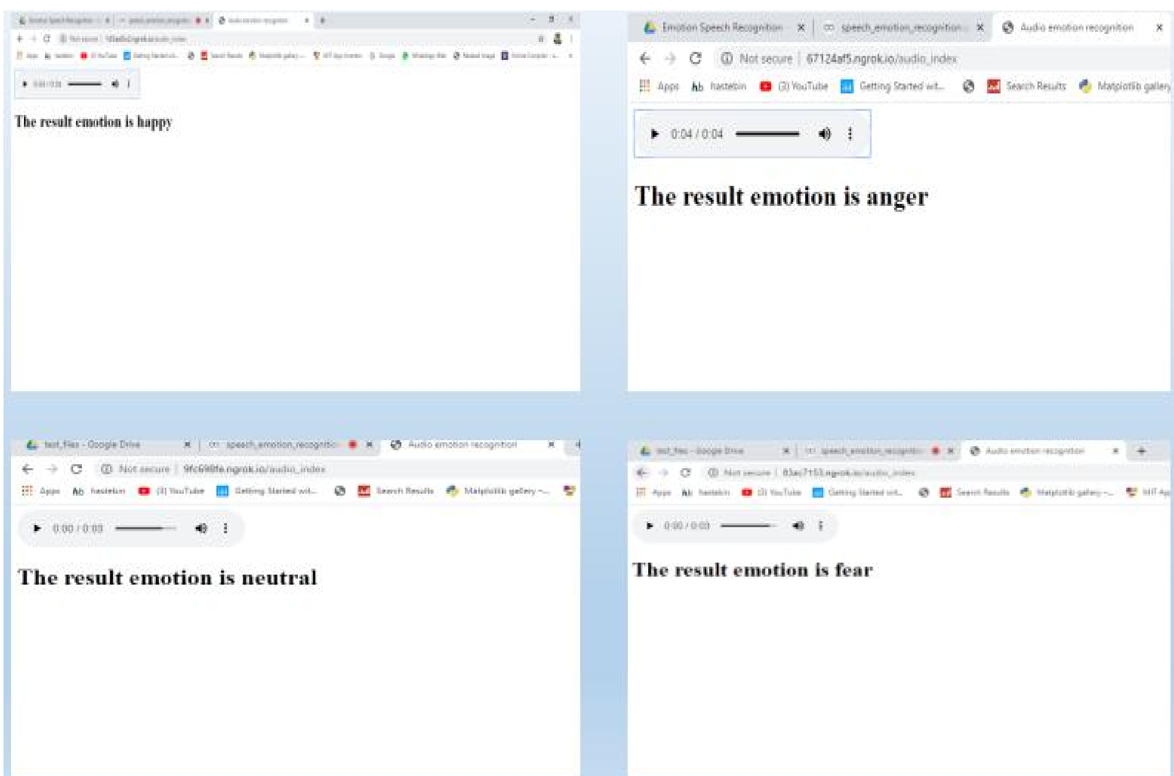
```
Epoch 83/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3919 - acc: 0.8549 - val_loss: 1.6001 - val_acc: 0.6059
Epoch 84/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.4080 - acc: 0.8493 - val_loss: 1.0905 - val_acc: 0.6543
Epoch 85/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3944 - acc: 0.8589 - val_loss: 0.8618 - val_acc: 0.6840
Epoch 86/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.4044 - acc: 0.8481 - val_loss: 0.9878 - val_acc: 0.6691
Epoch 87/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3839 - acc: 0.8592 - val_loss: 0.9427 - val_acc: 0.7138
Epoch 88/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3684 - acc: 0.8608 - val_loss: 1.2503 - val_acc: 0.6617
Epoch 89/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3541 - acc: 0.8772 - val_loss: 1.5294 - val_acc: 0.6320
Epoch 90/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3525 - acc: 0.8750 - val_loss: 0.9331 - val_acc: 0.6803
Epoch 91/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3497 - acc: 0.8744 - val_loss: 0.8997 - val_acc: 0.6952
Epoch 92/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3405 - acc: 0.8809 - val_loss: 0.8597 - val_acc: 0.7323
Epoch 93/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3108 - acc: 0.8893 - val_loss: 1.3313 - val_acc: 0.6245
Epoch 94/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3206 - acc: 0.8822 - val_loss: 1.2569 - val_acc: 0.6394
Epoch 95/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3014 - acc: 0.8890 - val_loss: 1.1308 - val_acc: 0.6877
Epoch 96/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3167 - acc: 0.8878 - val_loss: 1.0205 - val_acc: 0.7249
Epoch 97/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.2862 - acc: 0.9029 - val_loss: 0.9757 - val_acc: 0.6989
Epoch 98/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.2941 - acc: 0.8936 - val_loss: 1.3295 - val_acc: 0.6431
Epoch 99/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.2944 - acc: 0.8924 - val_loss: 1.4680 - val_acc: 0.6394
Epoch 100/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.2774 - acc: 0.9029 - val_loss: 1.2179 - val_acc: 0.6840
```

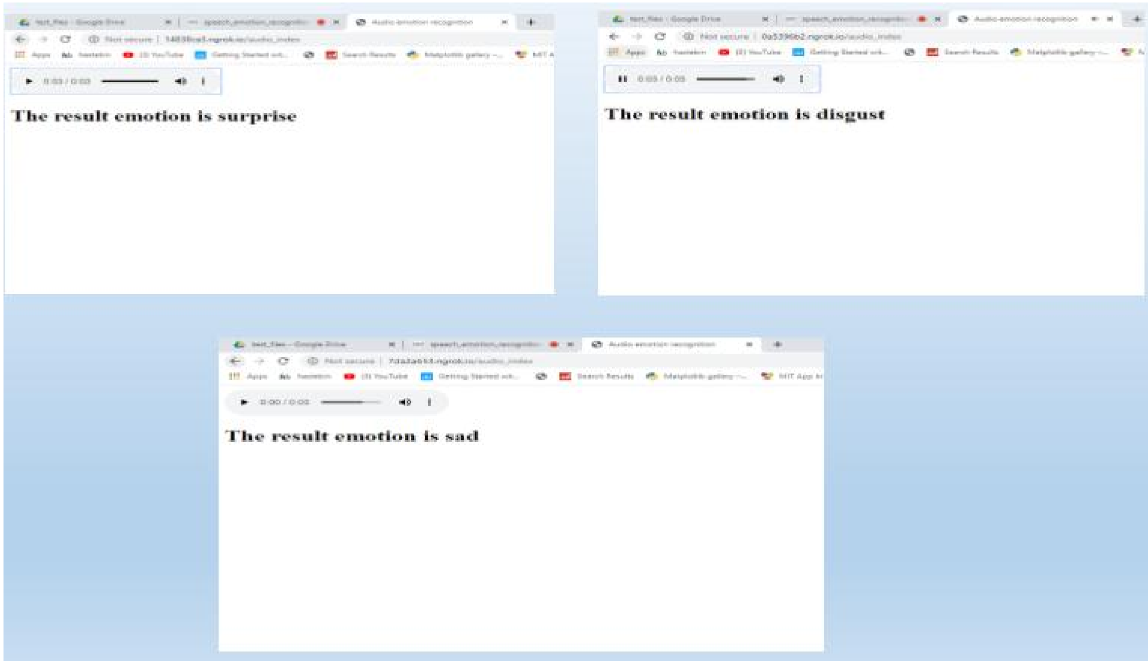
Output of Architecture:


Input_MELSPECT: InputLayer
input: [(?, 5, 128, 128, 1)]
output: [(?, 5, 128, 128, 1)]
Conv_1_MELSPECT(conv2d): TimeDistributed(Conv2D)
input: (?, 5, 128, 128, 1)
output: (?, 5, 128, 128, 64)
BatchNorm_1_MELSPECT(batch_normalization): TimeDistributed(BatchNormalization)
input: (?, 5, 128, 128, 64)
output: (?, 5, 128, 128, 64)
Activ_1_MELSPECT(activation): TimeDistributed(Activation)
input: (?, 5, 128, 128, 64)
output: (?, 5, 128, 128, 64)
MaxPool_1_MELSPECT(max_pooling2d): TimeDistributed(MaxPooling2D)
input: (?, 5, 128, 128, 64)
output: (?, 5, 64, 64, 64)
Drop_1_MELSPECT(dropout): TimeDistributed(Dropout)
input: (?, 5, 64, 64, 64)
output: (?, 5, 64, 64, 64)
Conv_2_MELSPECT(conv2d_1): TimeDistributed(Conv2D)
input: (?, 5, 64, 64, 64)
output: (?, 5, 64, 64, 64)
BatchNorm_2_MELSPECT(batch_normalization_1): TimeDistributed(BatchNormalization)
input: (?, 5, 64, 64, 64)
output: (?, 5, 64, 64, 64)
Activ_2_MELSPECT(activation_1): TimeDistributed(Activation)
input: (?, 5, 64, 64, 64)
output: (?, 5, 64, 64, 64)
MaxPool_2_MELSPECT(max_pooling2d_1): TimeDistributed(MaxPooling2D)
input: (?, 5, 64, 64, 64)
output: (?, 5, 16, 16, 64)
Drop_2_MELSPECT(dropout_1): TimeDistributed(Dropout)
input: (?, 5, 16, 16, 64)
output: (?, 5, 16, 16, 64)
Conv_3_MELSPECT(conv2d_2): TimeDistributed(Conv2D)
input: (?, 5, 16, 16, 64)
output: (?, 5, 16, 16, 128)
BatchNorm_3_MELSPECT(batch_normalization_2): TimeDistributed(BatchNormalization)
input: (?, 5, 16, 16, 128)
output: (?, 5, 16, 16, 128)
Activ_3_MELSPECT(activation_2): TimeDistributed(Activation)
input: (?, 5, 16, 16, 128)
output: (?, 5, 16, 16, 128)
MaxPool_3_MELSPECT(max_pooling2d_2): TimeDistributed(MaxPooling2D)
input: (?, 5, 16, 16, 128)
output: (?, 5, 4, 4, 128)
Drop_3_MELSPECT(dropout_2): TimeDistributed(Dropout)
input: (?, 5, 4, 4, 128)
output: (?, 5, 4, 4, 128)
Conv_4_MELSPECT(conv2d_3): TimeDistributed(Conv2D)
input: (?, 5, 4, 4, 128)
output: (?, 5, 4, 4, 128)
BatchNorm_4_MELSPECT(batch_normalization_3): TimeDistributed(BatchNormalization)
input: (?, 5, 4, 4, 128)
output: (?, 5, 4, 4, 128)
Activ_4_MELSPECT(activation_3): TimeDistributed(Activation)
input: (?, 5, 4, 4, 128)
output: (?, 5, 4, 4, 128)
MaxPool_4_MELSPECT(max_pooling2d_3): TimeDistributed(MaxPooling2D)
input: (?, 5, 4, 4, 128)
output: (?, 5, 1, 1, 128)
Drop_4_MELSPECT(dropout_3): TimeDistributed(Dropout)
input: (?, 5, 1, 1, 128)
output: (?, 5, 1, 1, 128)
Flat_MELSPECT(flatten): TimeDistributed(Flatten)
input: (?, 5, 1, 1, 128)
output: (?, 5, 128)
LSTM_1: LSTM
input: (?, 5, 128)
output: (?, 256)
FC: Dense
input: (?, 256)
output: (?, 7)

Output of Speech Emotion Recognition in Web page







6.1.2. Face Emotion Recognition

6.1.2.1. Pre-Processing

The models explored include :

- Manual filters
- Deep Learning Architectures
- DenseNet Inspired Architectures

```
### General imports ###
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from time import time
```

```
from time import sleep
import re
import os
import argparse
from collections import OrderedDict
import matplotlib.animation as animation
#Image processing
from scipy.ndimage import zoom
from scipy.spatial import distance
import imutils
from scipy import ndimage
import cv2
import dlib
from __future__ import division
from imutils import face_utils
#CNN models
%tensorflow_version 1.x
import keras
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_arra
y, load_img
from keras.callbacks import TensorBoard
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
```

```
from keras.layers.convolutional import Conv2D, MaxPooling2D, SeparableConv2D
from keras.utils import np_utils
from keras.regularizers import l2#, activity_l2
from keras.optimizers import SGD, RMSprop
from keras.utils import to_categorical
from keras.layers.normalization import BatchNormalization
from keras import models
from keras.utils.vis_utils import plot_model
from keras.layers import Input, GlobalAveragePooling2D
from keras.models import Model
from tensorflow.keras import layers
# Build SVM models
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import svm
#Same trained models
import h5py
from keras.models import model_from_json
import pickle
#Import Datas from Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

path = '/content/drive/My Drive/Colab Notebooks/'

local_path = '/content/drive/My Drive/Colab Notebooks/'

pd.options.mode.chained_assignment = None # default='warn' #to suppress SettingWith CopyWarning

#Reading the dataset

dataset = pd.read_csv(local_path + 'fer2013.csv')

#Obtaining train data where usage is "Training"

train = dataset[dataset["Usage"] == "Training"]

#Obtaining test data where usage is "PublicTest"

test = dataset[dataset["Usage"] == "PublicTest"]

#Converting " " separated pixel values to list

train['pixels'] = train['pixels'].apply(lambda image_px : np.fromstring(image_px, sep = ' '))

test['pixels'] = test['pixels'].apply(lambda image_px : np.fromstring(image_px, sep = ' '))

dataset.head()

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

dataset[dataset['emotion'] == 1].head()

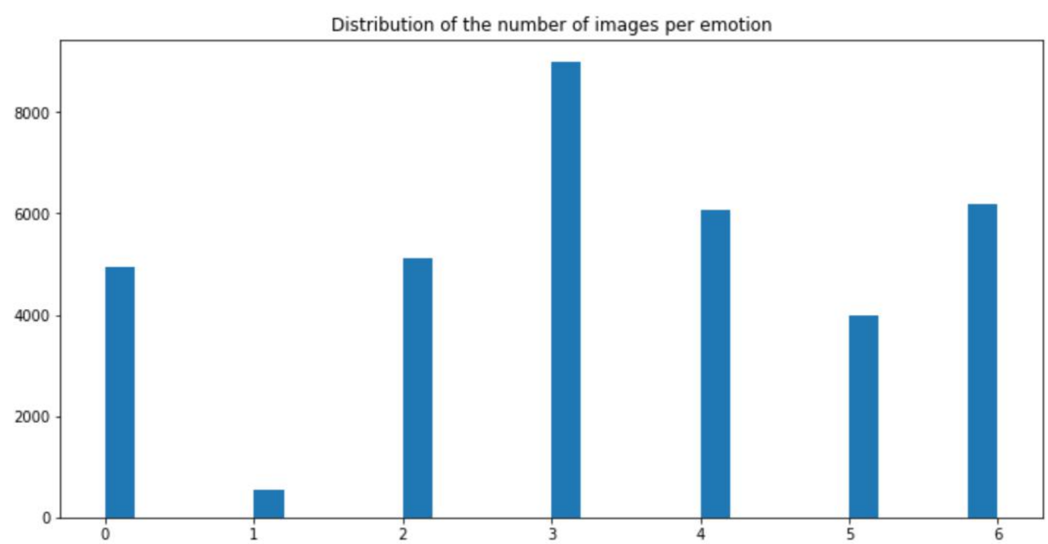
	emotion	pixels	Usage
299	1	126 126 129 120 110 168 174 172 173 174 170 15...	Training
388	1	89 55 24 40 43 48 53 55 59 41 33 31 22 32 42 4...	Training
416	1	204 195 181 131 50 50 57 56 66 98 138 161 173 ...	Training
473	1	14 11 13 12 41 95 113 112 111 122 132 137 142 ...	Training
533	1	18 25 49 75 89 97 100 100 101 103 105 107 107 ...	Training

plt.figure(figsize=(12,6))

plt.hist(dataset['emotion'], bins=30)

plt.title("Distribution of the number of images per emotion")

plt.show()



train.shape

test.shape

#Create the dataset

shape_x = 48

shape_y = 48

```
X_train = train.iloc[:, 1].values

y_train = train.iloc[:, 0].values

X_test = test.iloc[:, 1].values

y_test = test.iloc[:, 0].values

#np.vstack stack arrays in sequence vertically (picking element row wise)

X_train = np.vstack(X_train)

X_test = np.vstack(X_test)

#Reshape X_train, y_train,X_test,y_test in desired formats

X_train = np.reshape(X_train, (X_train.shape[0],48,48,1))

y_train = np.reshape(y_train, (y_train.shape[0],1))

X_test = np.reshape(X_test, (X_test.shape[0],48,48,1))

y_test = np.reshape(y_test, (y_test.shape[0],1))

print("Shape of X_train and y_train is " + str(X_train.shape) +" and " + str(y_train.shape)
 +" respectively.")

print("Shape of X_test and y_test is " + str(X_test.shape) +" and " + str(y_test.shape) +" r
espectively.")

# Change to float datatype

train_data = X_train.astype('float32')

test_data = X_test.astype('float32')

# Scale the data to lie between 0 to 1

train_data /= 255

test_data /= 255

# Change the labels from integer to categorical data
```

```
train_labels_one_hot = to_categorical(y_train)

test_labels_one_hot = to_categorical(y_test)

#Define the number of classes

# Find the unique numbers from the train labels

classes = np.unique(y_train)

nClasses = len(classes)

print('Total number of outputs : ', nClasses)

print('Output classes : ', classes)

# Find the shape of input images and create the variable input_shape

nRows,nCols,nDims = X_train.shape[1:]

input_shape = (nRows, nCols, nDims)

#Defining labels

def get_label(argument):

    labels = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Sad' , 5:'Surprise', 6:'Neutral'}

    return(labels.get(argument, "Invalid emotion"))

plt.figure(figsize=[10,5])

# Display the first image in training data

plt.subplot(121)

plt.imshow(np.squeeze(X_train[25,:,:], axis = 2), cmap='gray')

plt.title("Ground Truth : {}".format(get_label(int(y_train[25]))))


# Display the first image in testing data

plt.subplot(122)
```

```
plt.imshow(np.squeeze(X_test[26,:,:], axis = 2), cmap='gray')

plt.title("Ground Truth : {}".format(get_label(int(y_test[26]))))
```

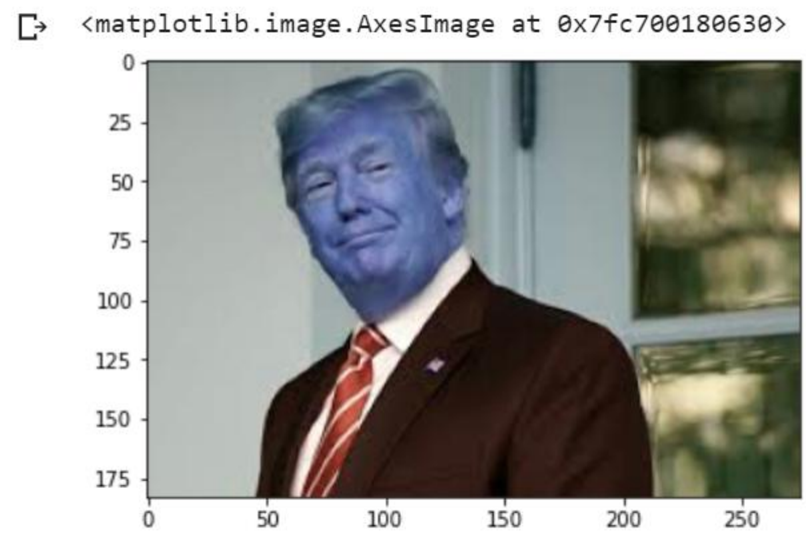
Detect Faces:

```
!pip install git+git://github.com/PnS2019/pnslib.git

from pnslib import utils

def detect_face(frame):

    #Cascade classifier pre-trained model

    #cascPath=files.upload()

    #cascPath = '/usr/local/lib/python3.7/site-
packages/cv2/data/haarcascade_frontalface_default.xml'

    #faceCascade = cv2.CascadeClassifier(cascPath)

    faceCascade = cv2.CascadeClassifier(

    utils.get_haarcascade_path('/content/drive/My Drive/Colab
Notebooks/haarcascade_frontalface_default.xml'))

#eye_cascade = cv2.CascadeClassifier(
```

```
#   utils.get_haarcascade_path('haarcascade_eye.xml'))


    #BGR -> Gray conversion
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


    #Cascade MultiScale classifier
   # detected_faces =
faceCascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=6,
    #                                       minSize=(shape_x, shape_y),
     #                                      flags=cv2.CASCADE_SCALE_IMAGE)
    detected_faces = faceCascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=6,
                                            minSize=(shape_x,shape_y),
                                            flags=cv2.CASCADE_SCALE_IMAGE)


    coord = []


    for x, y, w, h in detected_faces :
        if w > 100 :
            sub_img=frame[y:y+h,x:x+w]
            #cv2.rectangle(frame,(x,y),(x+w,y+h),(0, 255,255),1)
            coord.append([x,y,w,h])


    return gray, detected_faces, coord
```

```
#Extract facial features

def extract_face_features(faces, offset_coefficients=(0.075, 0.05)):
    gray = faces[0]
    detected_face = faces[1]

    new_face = []

    for det in detected_face :
        #Region in which the face is detected
        x, y, w, h = det
        #X and y correspond to the conversion to gray by gray, and w, h correspond to the
height / width

        #Offset coefficient, np.floor takes the lowest integer (delete border of the image)
        horizontal_offset = np.int(np.floor(offset_coefficients[0] * w))
        vertical_offset = np.int(np.floor(offset_coefficients[1] * h))

        #gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        #gray transforms the image
        extracted_face = gray[y+vertical_offset:y+h, x+horizontal_offset:x-
horizontal_offset+w]

        #Zoom on the extracted face
```

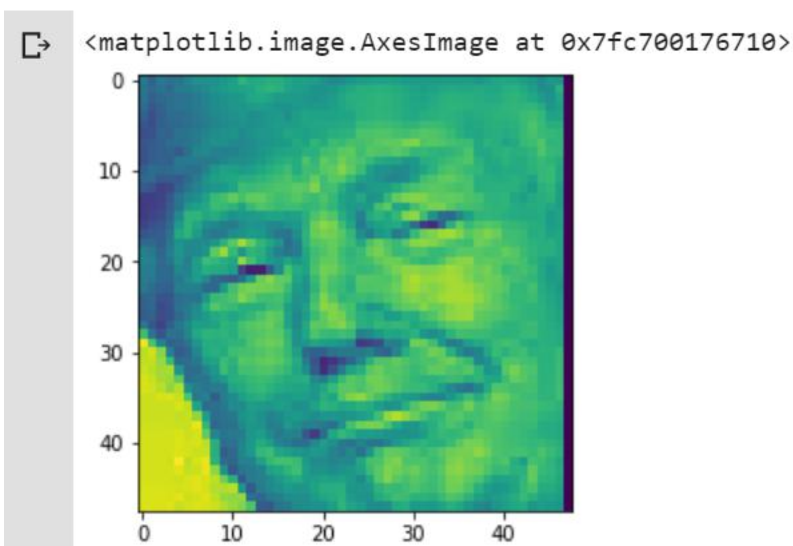
```
        new_extracted_face = zoom(extracted_face, (shape_x /
extracted_face.shape[0],shape_y / extracted_face.shape[1]))

        #cast type float

        new_extracted_face = new_extracted_face.astype(np.float32)

        #scale

        new_extracted_face /= float(new_extracted_face.max())

        #print(new_extracted_face)


        new_face.append(new_extracted_face)


    return new_face

from io import BytesIO

from PIL import Image

pic="/content/drive/My Drive/Colab Notebooks/test.jpg"

trump = Image.open(pic)

trump_face = cv2.imread(pic, cv2.COLOR_BGR2RGB)

plt.imshow(trump_face)
```



Extracted Face:

face = extract_face_features(detect_face(trump_face))[0]

plt.imshow(face)



6.1.2.2. Deep Learning Model Architectures (CNN)

A Simple Model:

def createModel():

```
#Model Initialization
model = Sequential()

#Adding Input Layer
model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=input_
shape))

#Adding more layers
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
    #Flattening
    model.add(Flatten())

    #Adding fully connected layer
    model.add(Dense(512, activation='relu'))

    model.add(Dropout(0.6))

    #Adding Output Layer
    model.add(Dense(nClasses, activation='softmax'))

    return model
```

Prevent Overfitting:

```
def createModel2():

    #Model Initialization
    model = Sequential()

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=input_
shape))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
```

```
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

    #Flattening
    model.add(Flatten())

    #Adding fully connected layer
    model.add(Dense(512, activation='relu'))
    #Adding Output Layer
    model.add(Dense(nClasses, activation='softmax'))
    return model
```

Deeper networks:

```
def createModel3():
    #Model Initialization
    model = Sequential()
```

```
model.add(Conv2D(20, (3, 3), padding='same', activation='relu',
input_shape=input_shape))
model.add(Conv2D(30, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(40, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(50, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(60, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(70, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(80, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(90, (3, 3), padding='same', activation='relu'))

#Flattening
model.add(Flatten())
```

```
#Adding fully connected layer

model.add(Dense(1000, activation='relu'))

model.add(Dense(512, activation='relu'))


#Adding Output Layer

model.add(Dense(nClasses, activation='softmax'))
```

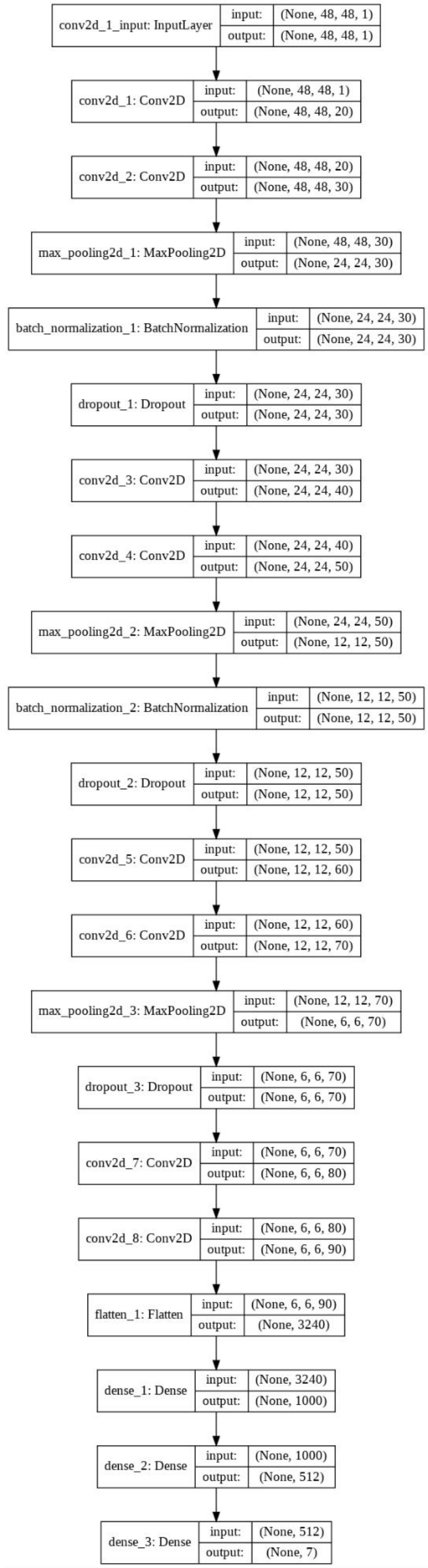
Build Model:

```
model = createModel3()

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 20)        200       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 30)        5430      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 30)        0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 30)        120       
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 30)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 40)        10840     
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 50)        18050     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 50)        0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 50)        200       
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 50)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 60)        27060     
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 70)        37870     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 70)          0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 70)          0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 80)          50480     
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 6, 6, 90)          64890     
_________________________________________________________________
flatten_1 (Flatten)          (None, 3240)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              3241000   
_________________________________________________________________
dense_2 (Dense)              (None, 512)               512512    
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 3591      
=================================================================
Total params: 3,972,243
Trainable params: 3,972,083
Non-trainable params: 160
_________________________________________________________________
```

And visualize the model architecture :

plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

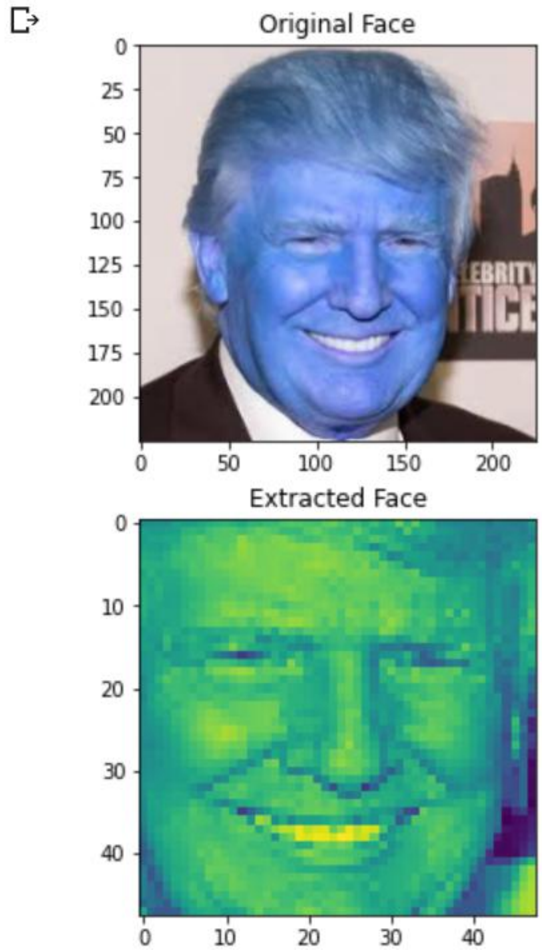


conv2d_1_input: InputLayer
input: (None, 48, 48, 1)
output: (None, 48, 48, 1)
conv2d_1: Conv2D
input: (None, 48, 48, 1)
output: (None, 48, 48, 20)
conv2d_2: Conv2D
input: (None, 48, 48, 20)
output: (None, 48, 48, 30)
max_pooling2d_1: MaxPooling2D
input: (None, 48, 48, 30)
output: (None, 24, 24, 30)
batch_normalization_1: BatchNormalization
input: (None, 24, 24, 30)
output: (None, 24, 24, 30)
dropout_1: Dropout
input: (None, 24, 24, 30)
output: (None, 24, 24, 30)
conv2d_3: Conv2D
input: (None, 24, 24, 30)
output: (None, 24, 24, 40)
conv2d_4: Conv2D
input: (None, 24, 24, 40)
output: (None, 24, 24, 50)
max_pooling2d_2: MaxPooling2D
input: (None, 24, 24, 50)
output: (None, 12, 12, 50)
batch_normalization_2: BatchNormalization
input: (None, 12, 12, 50)
output: (None, 12, 12, 50)
dropout_2: Dropout
input: (None, 12, 12, 50)
output: (None, 12, 12, 50)
conv2d_5: Conv2D
input: (None, 12, 12, 50)
output: (None, 12, 12, 60)
conv2d_6: Conv2D
input: (None, 12, 12, 60)
output: (None, 12, 12, 70)
max_pooling2d_3: MaxPooling2D
input: (None, 12, 12, 70)
output: (None, 6, 6, 70)
dropout_3: Dropout
input: (None, 6, 6, 70)
output: (None, 6, 6, 70)
conv2d_7: Conv2D
input: (None, 6, 6, 70)
output: (None, 6, 6, 80)
conv2d_8: Conv2D
input: (None, 6, 6, 80)
output: (None, 6, 6, 90)
flatten_1: Flatten
input: (None, 6, 6, 90)
output: (None, 3240)
dense_1: Dense
input: (None, 3240)
output: (None, 1000)
dense_2: Dense
input: (None, 1000)
output: (None, 512)
dense_3: Dense
input: (None, 512)
output: (None, 7)

Visualize layers and output

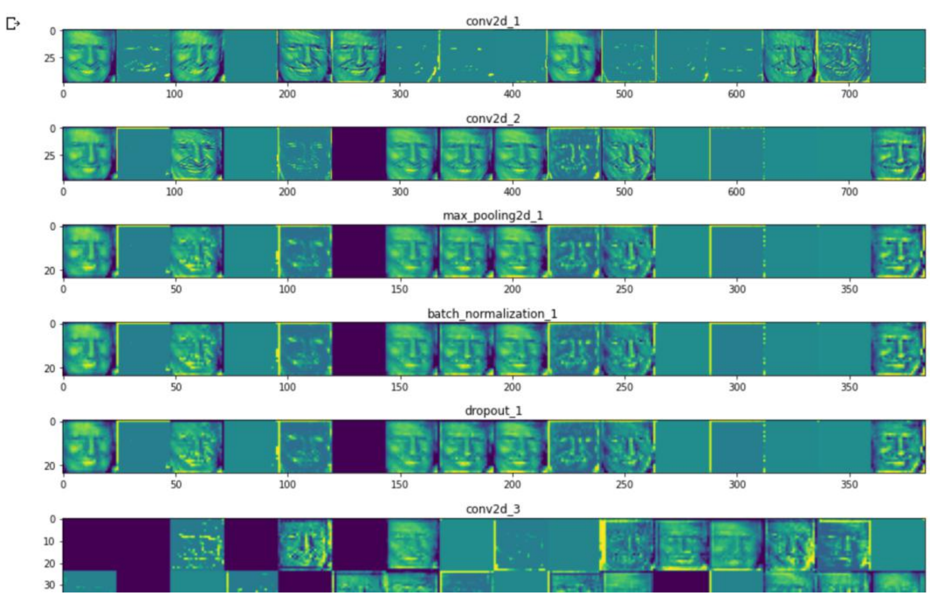
```
layer_outputs = [layer.output for layer in model.layers[:12]]

# Extracts the outputs of the top 12 layers

activation_model = models.Model(inputs=model.input, outputs=layer_outputs)

layer_names = []

for layer in model.layers[:12]:

   layer_names.append(layer.name)

 # Names of the layers

images_per_row = 16

pic='/content/drive/My Drive/Colab Notebooks/trump.png'

trump = Image.open(pic)

trump_face = cv2.imread("/content/drive/My Drive/Colab Notebooks/trump.png")

face = extract_face_features(detect_face(trump_face))[0]

to_predict = np.reshape(face.flatten(), (1,48,48,1))

res = model.predict(to_predict)

activations = activation_model.predict(to_predict)

plt.figure(figsize=(12,8))

plt.subplot(211)

plt.title("Original Face")

plt.imshow(trump_face)

plt.subplot(212)

plt.title("Extracted Face")

plt.imshow(face)
```

```
plt.show()
```




```
for layer_name, layer_activation in zip(layer_names, activations): # Displays the featur
e maps
    n_features = layer_activation.shape[-1] # Number of features in the feature map
    size = layer_activation.shape[1] #The feature map has shape (1, size, size, n_feature
s).
    n_cols = n_features // images_per_row # Tiles the activation channels in this matrix
    display_grid = np.zeros((size * n_cols, images_per_row * size))
    for col in range(n_cols): # Tiles each filter into a big horizontal grid
        for row in range(images_per_row):
```

```
            channel_image = layer_activation[0,:, :,col * images_per_row + row]
            channel_image -= channel_image.mean() # Post-
processes the feature to make it visually palatable
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size : (col + 1) * size, # Displays the grid
                         row * size : (row + 1) * size] = channel_image
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```



Create and train the model:

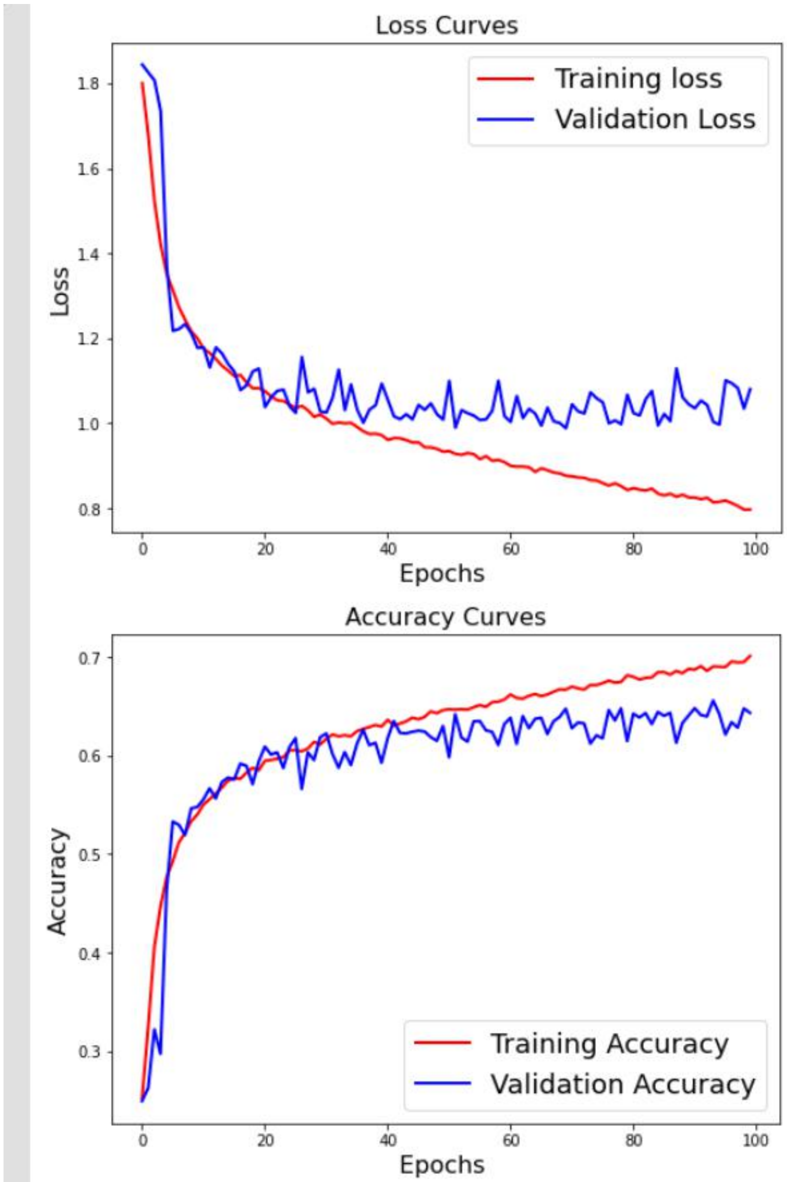
```
datagen = ImageDataGenerator(
        zoom_range=0.2,          # randomly zoom into images
        rotation_range=10,       # randomly rotate images in the range (degrees, 0 to 180)
        width_shift_range=0.1,   # randomly shift images horizontally (fraction of total wi
dth)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total heig
ht)
        horizontal_flip=True,    # randomly flip images
        vertical_flip=False)     # randomly flip images
#Creating 2nd model and training(fitting)


model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


batch_size = 256
epochs = 100


# Fit the model on the batches generated by datagen.flow().
history = model.fit_generator(
    datagen.flow(train_data, train_labels_one_hot, batch_size=batch_size),
    steps_per_epoch=int(np.ceil(train_data.shape[0] / float(batch_size))),
    epochs = epochs,
    validation_data=(test_data, test_labels_one_hot)
```

```
Epoch 90/100
113/113 [==============================] - 17s 149ms/step - loss: 0.8281 - accuracy: 0.6879 - val_loss: 1.0447 - val_accuracy: 0.6406
Epoch 91/100
113/113 [==============================] - 17s 149ms/step - loss: 0.8258 - accuracy: 0.6872 - val_loss: 1.0355 - val_accuracy: 0.6481
Epoch 92/100
113/113 [==============================] - 17s 149ms/step - loss: 0.8203 - accuracy: 0.6908 - val_loss: 1.0535 - val_accuracy: 0.6411
Epoch 93/100
113/113 [==============================] - 17s 151ms/step - loss: 0.8261 - accuracy: 0.6859 - val_loss: 1.0429 - val_accuracy: 0.6397
Epoch 94/100
113/113 [==============================] - 17s 150ms/step - loss: 0.8147 - accuracy: 0.6904 - val_loss: 1.0034 - val_accuracy: 0.6559
Epoch 95/100
113/113 [==============================] - 17s 151ms/step - loss: 0.8147 - accuracy: 0.6901 - val_loss: 0.9970 - val_accuracy: 0.6422
Epoch 96/100
113/113 [==============================] - 17s 151ms/step - loss: 0.8186 - accuracy: 0.6899 - val_loss: 1.1012 - val_accuracy: 0.6213
Epoch 97/100
113/113 [==============================] - 17s 151ms/step - loss: 0.8140 - accuracy: 0.6954 - val_loss: 1.0943 - val_accuracy: 0.6339
Epoch 98/100
113/113 [==============================] - 17s 150ms/step - loss: 0.8044 - accuracy: 0.6945 - val_loss: 1.0828 - val_accuracy: 0.6283
Epoch 99/100
113/113 [==============================] - 17s 150ms/step - loss: 0.7973 - accuracy: 0.6948 - val_loss: 1.0347 - val_accuracy: 0.6478
Epoch 100/100
113/113 [==============================] - 17s 150ms/step - loss: 0.7962 - accuracy: 0.7010 - val_loss: 1.0804 - val_accuracy: 0.6434
```

Evaluate the model:

```
#Plotting accuracy and loss curves for 2nd model


# Loss Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=2.0)
plt.plot(history.history['val_loss'],'b',linewidth=2.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)


# Accuracy Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['acc'],'r',linewidth=2.0)
plt.plot(history.history['val_acc'],'b',linewidth=2.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
```

```
plt.xlabel('Epochs ',fontsize=16)

plt.ylabel('Accuracy',fontsize=16)

plt.title('Accuracy Curves',fontsize=16)
```



Save and reopen the model:

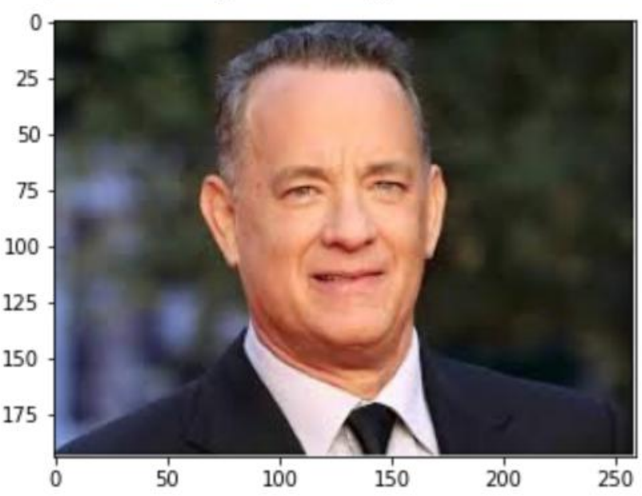
```
#save the model weights

json_string = model.to_json()

model.save_weights('/content/drive/My Drive/Colab Notebooks/model_3.h5')
```

```
open('/content/drive/My Drive/Colab Notebooks/model_3.json', 'w').write(json_string)

model.save_weights(local_path + 'savedmodels/Emotion_Face_Detection_Model.h5')

with open('/content/drive/My Drive/Colab Notebooks/model_3.json','r') as f:
    json = f.read()

model = model_from_json(json)

model.load_weights('/content/drive/My Drive/Colab Notebooks/model_3.h5')

model.save('/content/drive/My Drive/Colab Notebooks/model_fer.h5')

print("Loaded model from disk")
```

Making prediction on an image:

```
pic='/content/drive/My Drive/Colab Notebooks/hanks.png'

hanks = Image.open(pic)

hanks_face = cv2.imread(pic)

plt.imshow(cv2.cvtColor(hanks_face, cv2.COLOR_BGR2RGB))
```

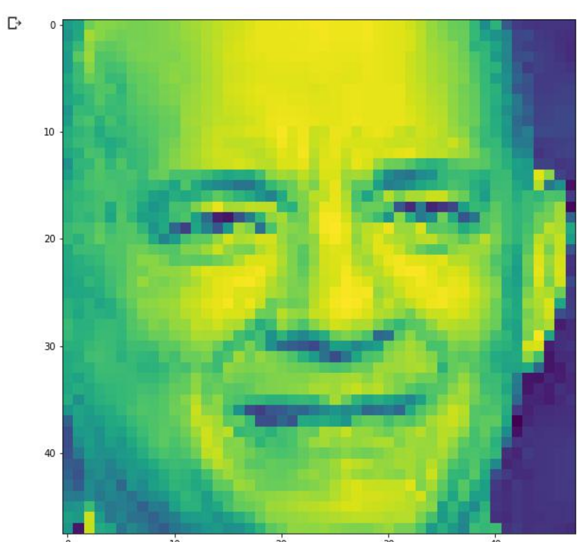
```
<matplotlib.image.AxesImage at 0x7fc6a2089470>
```




```
plt.figure(figsize=(12,12))

plt.imshow(detect_face(hanks_face)[0])
```

```
plt.show()
```


```
for face in extract_face_features(detect_face(hanks_face)) :
    plt.figure(figsize=(10,10))
    plt.imshow(face)
    plt.show()
```




```
for face in extract_face_features(detect_face(hanks_face)) :
    to_predict = np.reshape(face.flatten(), (1,48,48,1))
```

```
    res = model.predict(to_predict)

    result_num = np.argmax(res)

    print(result_num)
```

```
3
```

This corresponds to the Happy Labels which is a good prediction.

Enhanced visualization:

Frequency of eye bink:

```
def eye_aspect_ratio(eye):
  A = distance.euclidean(eye[1], eye[5])
  B = distance.euclidean(eye[2], eye[4])
  C = distance.euclidean(eye[0], eye[3])
  ear = (A + B) / (2.0 * C)
  return ear
thresh = 0.25
frame_check = 20
face_detect = dlib.get_frontal_face_detector()
predictor_landmarks = dlib.shape_predictor(local_path+"shape_predictor_68_face_landmarks.dat")
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

Detect Keypoints to plot them:

```
(nStart, nEnd) = face_utils.FACIAL_LANDMARKS_IDXS["nose"]
```

```
(mStart, mEnd) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]
(jStart, jEnd) = face_utils.FACIAL_LANDMARKS_IDXS["jaw"]
(eblStart, eblEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eyebrow"]
(ebrStart, ebrEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eyebrow"]
```

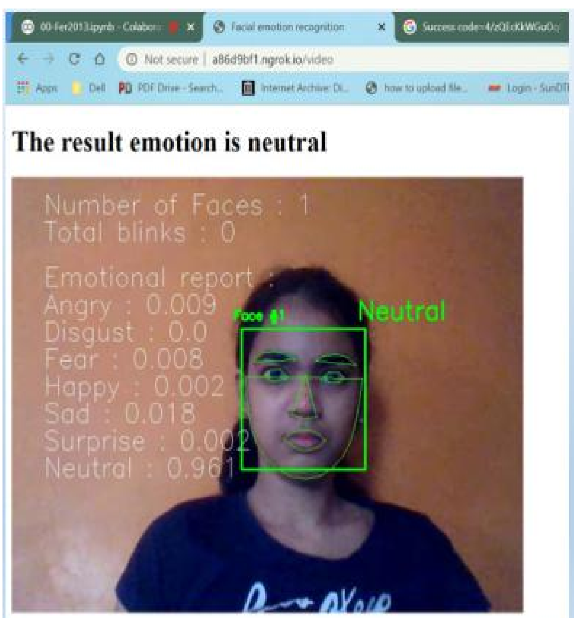
Face Alignment:

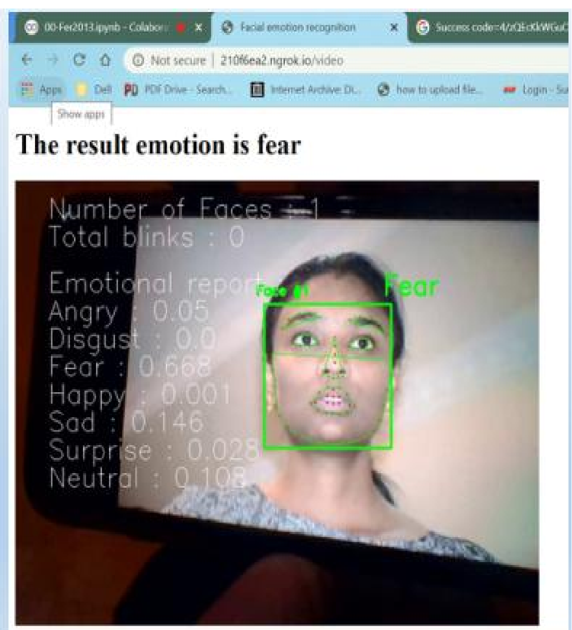
```
desiredLeftEye=(0.35, 0.35)


def align(gray, rect):
    # convert the landmark (x, y)-coordinates to a NumPy array
    shape = predictor(gray, rect)
    shape = shape_to_np(shape)


    # extract the left and right eye (x, y)-coordinates
    (lStart, lEnd) = FACIAL_LANDMARKS_IDXS["left_eye"]
    (rStart, rEnd) = FACIAL_LANDMARKS_IDXS["right_eye"]
    leftEyePts = shape[lStart:lEnd]
    rightEyePts = shape[rStart:rEnd]


    # compute the center of mass for each eye
    leftEyeCenter = leftEyePts.mean(axis=0).astype("int")
    rightEyeCenter = rightEyePts.mean(axis=0).astype("int")


    # compute the angle between the eye centroids
```

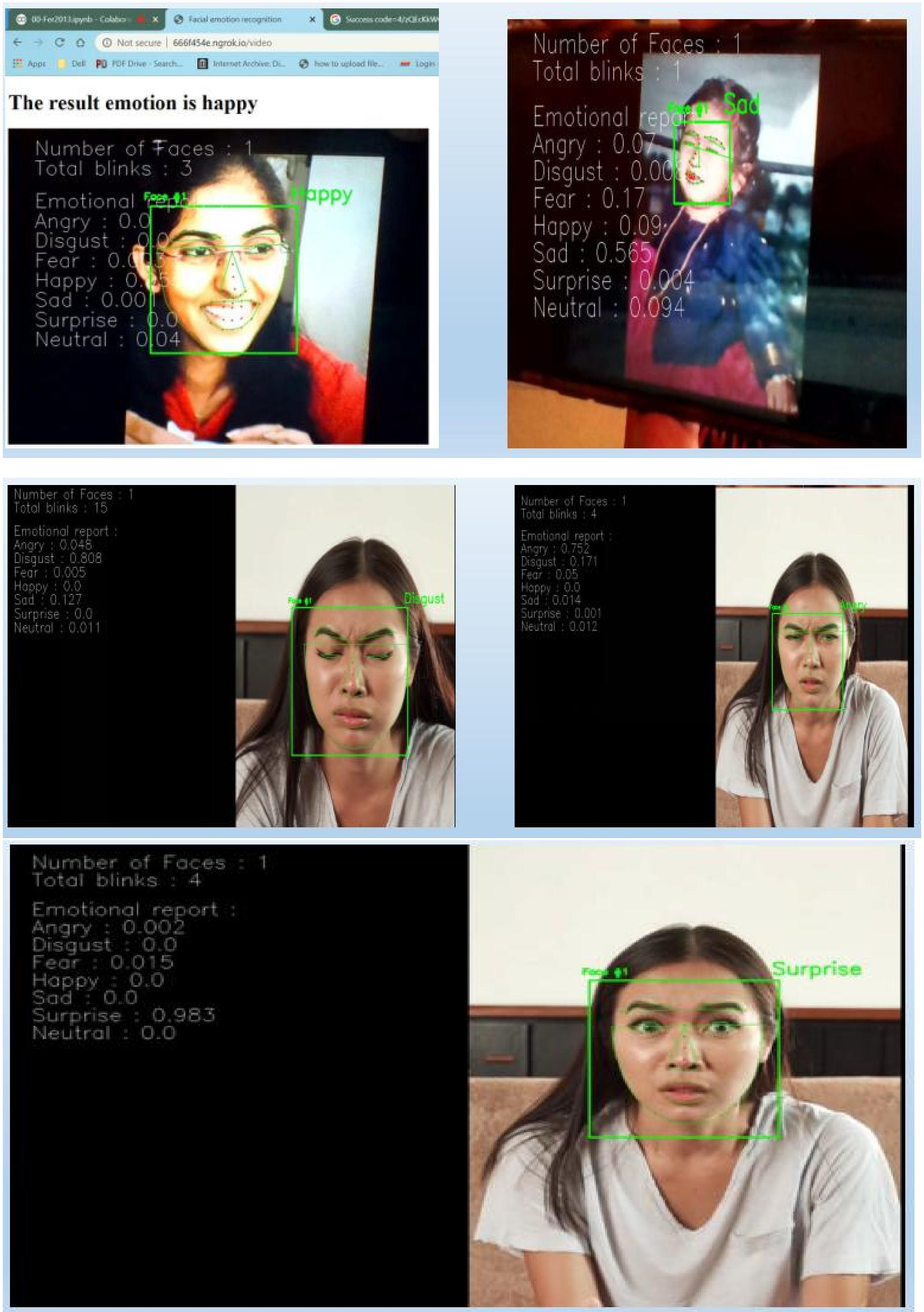
```
dY = rightEyeCenter[1] - leftEyeCenter[1]
dX = rightEyeCenter[0] - leftEyeCenter[0]
angle = np.degrees(np.arctan2(dY, dX)) - 180

# compute the desired right eye x-coordinate based on the
# desired x-coordinate of the left eye
desiredRightEyeX = 1.0 - desiredLeftEye[0]

# determine the scale of the new resulting image by taking
# the ratio of the distance between eyes in the *current*
# image to the ratio of distance between eyes in the
# *desired* image
dist = np.sqrt((dX ** 2) + (dY ** 2))
desiredDist = (desiredRightEyeX - desiredLeftEye[0])
desiredDist *= self.desiredFaceWidth
scale = desiredDist / dist

# compute center (x, y)-coordinates (i.e., the median point)
# between the two eyes in the input image
eyesCenter = ((leftEyeCenter[0] + rightEyeCenter[0]) // 2,
    (leftEyeCenter[1] + rightEyeCenter[1]) // 2)

# grab the rotation matrix for rotating and scaling the face
```

```
M = cv2.getRotationMatrix2D(eyesCenter, angle, scale)


# update the translation component of the matrix
tX = self.desiredFaceWidth * 0.5
tY = self.desiredFaceHeight * self.desiredLeftEye[1]
M[0, 2] += (tX - eyesCenter[0])
M[1, 2] += (tY - eyesCenter[1])


# apply the affine transformation
(w, h) = (self.desiredFaceWidth, self.desiredFaceHeight)
#output = cv2.warpAffine(image, M, (w, h), flags=cv2.INTER_CUBIC)
output = cv2.warpAffine(gray, M, (w, h), flags=cv2.INTER_CUBIC)
# return the aligned face
return output
```

FACE EMOTION RECOGNITION RESULT IN WEBPAGE






Not secure | 666f454e.ngrok.io/video
The result emotion is happy
Number of Faces : 1
Total blinks : 3
Happy
Number of Faces : 1
Sad
Fear : 0.17
Happy : 0.09
Sad : 0.565
Surprise : 0.004
Neutral : 0.094
Number of Faces : 1
Total blinks : 15
Emotional report :
Angry : 0.048
Disgust : 0.808
Fear : 0.005
Happy : 0.0
Sad : 0.127
Surprise : 0.0
Neutral : 0.011
Disgust
Number of Faces : 1
Total blinks : 4
Emotional report :
Angry : 0.752
Disgust : 0.171
Fear : 0.05
Happy : 0.0
Sad : 0.014
Surprise : 0.001
Neutral : 0.012
Angry
Number of Faces : 1
Total blinks : 4
Emotional report :
Angry : 0.002
Disgust : 0.0
Fear : 0.015
Happy : 0.0
Sad : 0.0
Surprise : 0.983
Neutral : 0.0
Face #1
Surprise

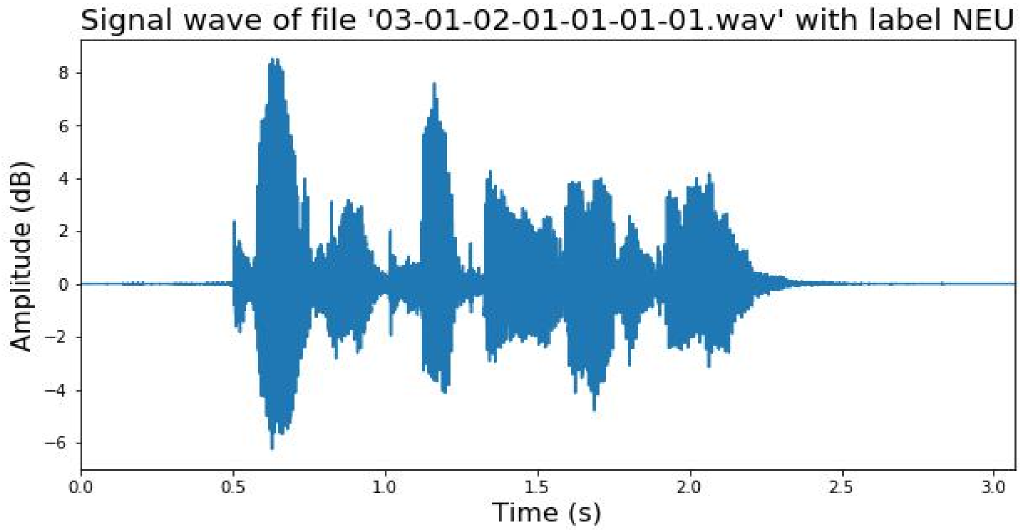
CHAPTER 7
TESTING

7.1.Speech Emotion Recognition:

Testing is basically, evaluating the accuracy of a model, when speech emotion recognition model is tested, we have achieved an accuracy of about 96% on training and about 68% on validation.

```
Epoch 83/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3919 - acc: 0.8549 - val_loss: 1.6001 - val_acc: 0.6059
Epoch 84/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.4080 - acc: 0.8493 - val_loss: 1.0905 - val_acc: 0.6543
Epoch 85/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3944 - acc: 0.8589 - val_loss: 0.8618 - val_acc: 0.6840
Epoch 86/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.4044 - acc: 0.8481 - val_loss: 0.9878 - val_acc: 0.6691
Epoch 87/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3839 - acc: 0.8592 - val_loss: 0.9427 - val_acc: 0.7138
Epoch 88/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3684 - acc: 0.8608 - val_loss: 1.2503 - val_acc: 0.6617
Epoch 89/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3541 - acc: 0.8772 - val_loss: 1.5294 - val_acc: 0.6320
Epoch 90/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3525 - acc: 0.8750 - val_loss: 0.9331 - val_acc: 0.6803
Epoch 91/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3497 - acc: 0.8744 - val_loss: 0.8997 - val_acc: 0.6952
Epoch 92/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3405 - acc: 0.8809 - val_loss: 0.8597 - val_acc: 0.7323
Epoch 93/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3108 - acc: 0.8893 - val_loss: 1.3313 - val_acc: 0.6245
Epoch 94/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3206 - acc: 0.8822 - val_loss: 1.2569 - val_acc: 0.6394
Epoch 95/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3014 - acc: 0.8890 - val_loss: 1.1308 - val_acc: 0.6877
Epoch 96/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.3167 - acc: 0.8878 - val_loss: 1.0205 - val_acc: 0.7249
Epoch 97/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.2862 - acc: 0.9029 - val_loss: 0.9757 - val_acc: 0.6989
Epoch 98/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.2941 - acc: 0.8936 - val_loss: 1.3295 - val_acc: 0.6431
Epoch 99/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.2944 - acc: 0.8924 - val_loss: 1.4680 - val_acc: 0.6394
Epoch 100/100
3225/3225 [==============================] - 28s 9ms/sample - loss: 0.2774 - acc: 0.9029 - val_loss: 1.2179 - val_acc: 0.6840
```

```
1344
4
NEU
[-0.00020196 -0.00020196 -0.00020196 ... -0.00020215 -0.00020169
 -0.00020233]
03-01-02-01-01-01-01.wav
```

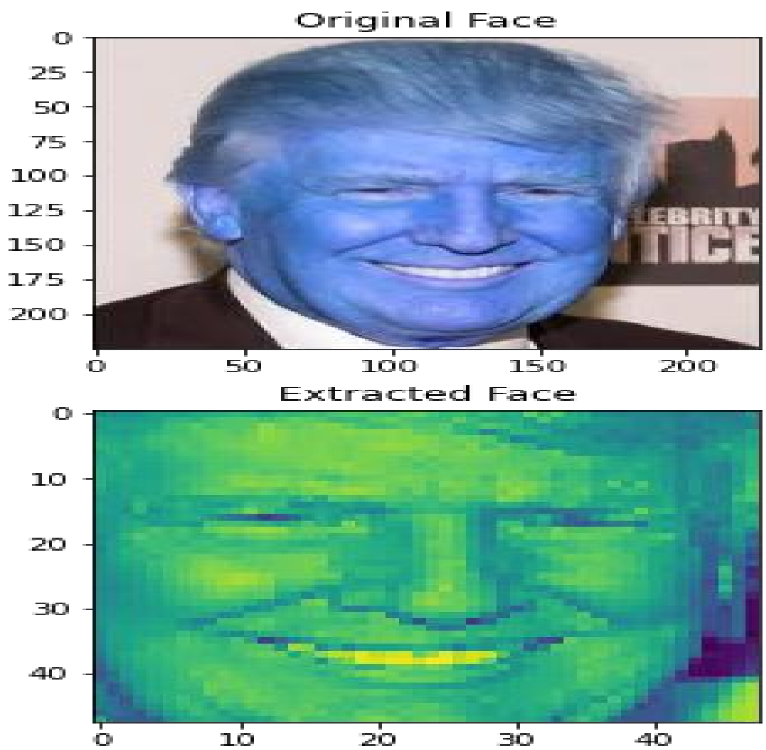



```
Audio file '03-01-02-01-01-01-01.wav':
```

7.2. Face Emotion Recognition:

For face emotion recognition model, we have achieved an accuracy of about 89% on training and about 72% on validation.

```
Epoch 90/100
113/113 [==============================] - 17s 149ms/step - loss: 0.8281 - accuracy: 0.6879 - val_loss: 1.0447 - val_accuracy: 0.6406
Epoch 91/100
113/113 [==============================] - 17s 149ms/step - loss: 0.8258 - accuracy: 0.6872 - val_loss: 1.0355 - val_accuracy: 0.6481
Epoch 92/100
113/113 [==============================] - 17s 149ms/step - loss: 0.8203 - accuracy: 0.6908 - val_loss: 1.0535 - val_accuracy: 0.6411
Epoch 93/100
113/113 [==============================] - 17s 151ms/step - loss: 0.8261 - accuracy: 0.6859 - val_loss: 1.0429 - val_accuracy: 0.6397
Epoch 94/100
113/113 [==============================] - 17s 150ms/step - loss: 0.8147 - accuracy: 0.6904 - val_loss: 1.0034 - val_accuracy: 0.6559
Epoch 95/100
113/113 [==============================] - 17s 151ms/step - loss: 0.8147 - accuracy: 0.6901 - val_loss: 0.9970 - val_accuracy: 0.6422
Epoch 96/100
113/113 [==============================] - 17s 151ms/step - loss: 0.8186 - accuracy: 0.6899 - val_loss: 1.1012 - val_accuracy: 0.6213
Epoch 97/100
113/113 [==============================] - 17s 151ms/step - loss: 0.8140 - accuracy: 0.6954 - val_loss: 1.0943 - val_accuracy: 0.6339
Epoch 98/100
113/113 [==============================] - 17s 150ms/step - loss: 0.8044 - accuracy: 0.6945 - val_loss: 1.0828 - val_accuracy: 0.6283
Epoch 99/100
113/113 [==============================] - 17s 150ms/step - loss: 0.7973 - accuracy: 0.6948 - val_loss: 1.0347 - val_accuracy: 0.6478
Epoch 100/100
113/113 [==============================] - 17s 150ms/step - loss: 0.7962 - accuracy: 0.7010 - val_loss: 1.0804 - val_accuracy: 0.6434
```

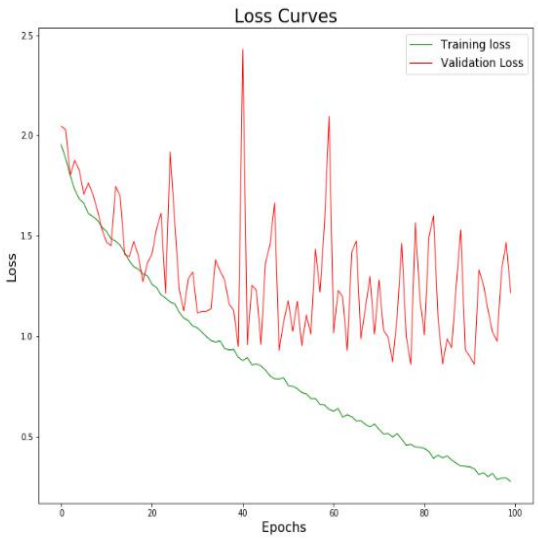

Original Face
0
25
50
75
100
125
150
175
200
0
50
100
150
200
Extracted Face
0
10
20
30
40
0
10
20
30
40

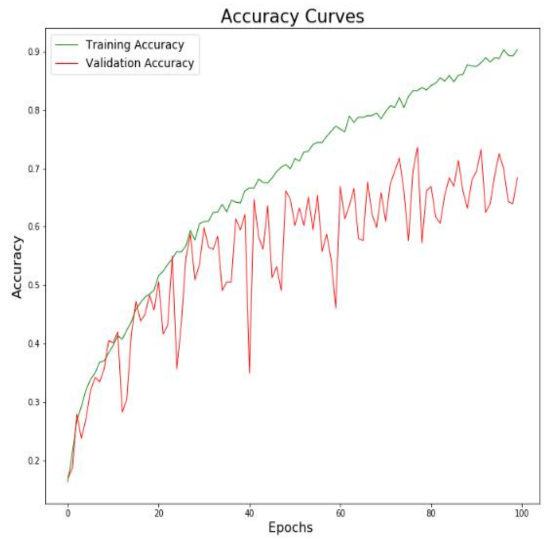
CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Conclusion:

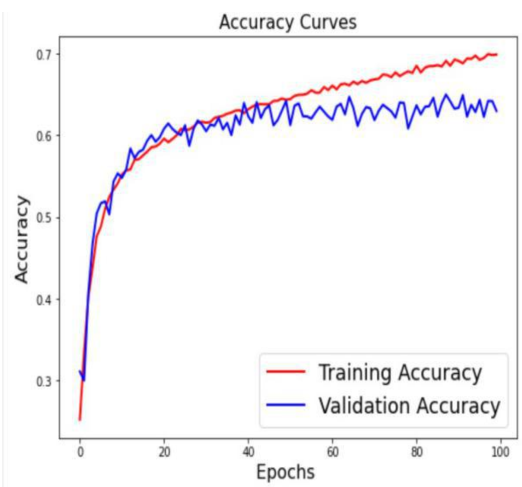
Most of the current research concentrate on investigating different features and their correlation with emotional state in audio, video and text. In this fact some researcher develop their own feature like MLS to achieve high performance in recognition rate. In most of them, it is hard even for human to specify different emotion of certain collected utterances. In conclusion, there are only limited studies that considered applying multiple classifier to speech emotion recognition. We reviewed and discussed various emotional recognition systems based approaches. We also compare its performance in terms of classifier, features, recognition rate, and datasets. Well-design classifiers have obtain high classification accuracies between different types of emotions. In this current approach, we presented an automatic multimodal emotion recognition (MER) system using CNN as the machine learning algorithm which classify 7 emotions. Thus, We used Ravdess dataset for speech using time distributed CNN which extracts MFCC, Mel-Scale, Spectrogram features. For video, Fer2013 dataset using CNN model with synthesized features like haar features, HOG sliding windows, HOG features and Facial landmarks are used. Infact, we study how classifiers and features impact recognition accuracy of emotions in audio and video. A subset of highly discriminant features is selected. Feature selection techniques show that more information is not always good in machine learning applications. The machine learning models were trained and evaluated to recognize emotional states from these features. When we implemented deep networks on ravdess dataset, we acheived an accuracy of 96% on training set and 78% of accuracy on testing set and about 89% accuracy on training and 79% on validation when face emotion is recognized using Fer2013 dataset and features like haar features, HOG sliding windows, HOG features and Facial landmarks are used. From this result, we can see that CNN often perform better with more data and it suffers from the problem of very long training times.

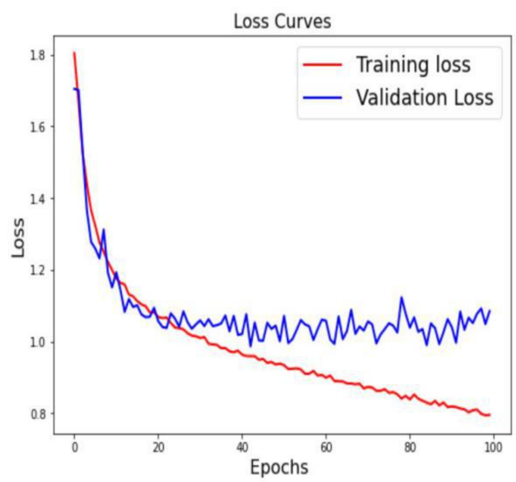
Speech Emotion Recognition:





Face Emotion Recognition:





8.2 Future Scope:

1. In future, we can do fusion of audio, video and text emotion recognition models using different models.
2. We can integrate this system in several applications where emotion plays an important role.
3. The suicidal rate of people can also be decreased when these applications can find the emotional status (like depressed state) of the person well before.

CHAPTER 9
BIBILOGRAPHY

[1] Seunghyun Yoon, Seokhyun Byun, and Kyomin Jung, Multimodal speech emotion recognition using audio and text. Accepted as a conference paper at IEEE SLT 2018.

[2] Samarth Tripathi, Homayoon Beigi, Multi-modal emotion recognition on IEMOCAP with neural networks. Submitted on (arXiv:1804.05788) 16 Apr 2018.

[3] Gaurav Sahu, Multimodal speech emotion recognition and ambiguity resolution. In arXiv:1904.06022, Submitted on 12 Apr 2019.

[4] N. Majumdera, D. Hazarikab, A. Gelbukha, E. Cambriac, S. Poriac, Multimodal sentiment analysis using hierarchical fusion with context modeling. Accepted 28 July 2018, Elsevier.

[5] Zheng Lian, Ya Li, Jianhua Tao and Jian Huang, Investigation of multimodal features, classifiers and fusion methods for emotion recognition. Published on September 2018.

[6] Nusrat j. Shoumy, li-minn ang, Kah Phooi Seng, d.m.Motiur Rahaman, Tanveer Zia, Multimodal big data affective analytics: a comprehensive survey using text, audio, visual and physiological signals.

[7] Jose M. Arana, Fernando Gordillo, Jeannete Darias, Lilia Mestas, Analysis of the efficacy and reliability of the moodies app for detecting emotions through speech: does it actually work?

[8] Minjie Ren, Weizhi Nie, Anan Liu, Yuting Su, Multi-modal correlated network for emotion recognition in speech. In visual Informatics, Volume 3, Issue 3, September 2019.

[9] Maryam Imani, Gholam Ali Montazer, A survey of emotion recognition methods with emphasis on e-learning environments. Published in J. Netw. Comput. Appl. 2019.

[10] Yingying Jiang, Wei Li, M. Shamim Hossain, Min Chen, Abdulhameed Alelaiwi, Muneer Al-Hammadi, A snapshot research and implementation of multimodal

information fusion for data-driven emotion recognition. In Elsevier, Accepted on 9 June 2019.

[11] S. Lalitha, Shikha Tripathi, Deepa Gupta, Enhanced speech emotion detection using deep neural networks. In International Journal of Speech Technology, Issues on march,2019.

[12] Mumtaz Begum Mustafa, Mansoor A. M. Yusoof, Zuraidah M. Don, Mehdi Malekzadeh, Speech emotion recognition research: an analysis of research focus. In International Journal of Speech Technology, Issued on Jan, 2018.

[13] Nicholas Cummins, Stefan Scherer, Jarek Krajewski, Sebastian Schnieder, Julien Epps, Thomas F. Quatieri, A review of depression and suicide risk assessment using speech a