Emotion Recognition from Video

Human beings display their emotions using facial expressions. For humans it is very easy to recognize
those emotions but for computers it is very challenging. Facial expressions vary from person to person. Brightness, contrast and resolution of every random image is different. This is why recognizing facial expressions is very difficult. Facial expression recognition is an active research area. In this project, we worked on recognition of seven basic human emotions. These emotions are angry, disgust, fear, happy, sad, surprise and neutral.
The model involves generally 3 steps training, testing and validation and after completion of these steps we input our audio or video to predict emotion in the input.

FER2013 Dataset-

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories .
(0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).
The training set consists of 28,709 examples. The public test set consists of 3,589 examples.

Firstly we have done emotion recognition using video in which we used basic CNN algorithm , trained and tested with the help of fer2013 dataset to predict emotion. Below you can see the code and explanation of the video part. Emotion recognition using video has been completed. Now coming to emotion recognition using the audio part , RAVDESS dataset https://www.kaggle.com/uwrfkaggler/ravdess-emotional-speech-audio will be used in this part. We have started studying RNN and research papers of deep speech to implement this part. And then we will try to sum up both parts and predict emotion using audio and video both.

Import required libraries:

Import the required libraries for building the network. The code for importing the libraries is written below.

```
import pandas as pd
import  sys,os
import numpy as np import keras
from  keras.models import  Sequential
from  keras.layers.core  import  Dropout
from  keras.layers  import  Dense,  Conv2D,  MaxPooling2D  ,  Flatten
from  tensorflow.keras.layers.experimental  import  preprocessing
from  keras.optimizers  import  SGD
import cv2 import np_utils
from  keras.preprocessing  import  image
from  keras.preprocessing.image  import  ImageDataGenerator
```

After importing all the libraries we have to import dataset which is to be used,we can give the directory in small brackets where we have stored the dataset as shown in the code below.

```
emotion_data = pd.read_csv('C:\\Users\\HP\\Desktop\\fer2013.csv')
```

We will then create a different list for testing and training image pixels. After this we will check if pixels belong to the training set or testing set and we will append to the training or testing list respectively.The code for this is written below.

```
X_train = [] y_train = [] X_test
= [] y_test = []
for index, row in emotion_data.iterrows():
    k = row['pixels'].split(" ")

    if row['Usage'] == 'Training':

        X_train.append(np.array(k))
        y_train.append(row['emotion'])
    elif row['Usage'] == 'PublicTest':
        X_test.append(np.array(k))
        y_test.append(row['emotion'])
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test) #print(X_train.shape)
#print(X_test.shape) #print(y_test.shape)
#print(y_test.shape)

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

X_test = X_test.reshape(X_test.shape[0], 48, 48, 1)
#print(X_train.shape)
y_train= keras.utils.to_categorical(y_train, num_classes=7)
```

Now it's time to design the CNN model for emotion detection with different layers. We start with the initialization of the model followed by batch normalization layer and then different convents layers with ReLu as an activation function, max pool layers, and dropouts to do learning efficiently.

```
model = Sequential()
#1st 1ayer
model.add(Conv2D(64,(5,5),activation='relu',input_shape=(48,48,1)))
preprocessing.RandomFlip(mode='horizontal')
preprocessing.RandomRotation(factor=0.05)
model.add(MaxPooling2D(pool_size=(5,5),strides=(2,2)))
#2nd 1ayer
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
#3rd 1ayer
 model.add(Conv2D(256,(3,3),activation='relu'))
model.add(Conv2D(256,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(1024,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(7,activation='softmax')) gen =
ImageDataGenerator()
train_gen = gen.flow(X_train,y_train,batch_size=512)
gen1= ImageDataGenerator()
test_gen=gen1.flow(X_test,y_test,batch_size=512)
```

We have also compiled the model using Adam as an optimizer, loss as categorical cross-entropy, and metrics as accuracy as shown in the below code.

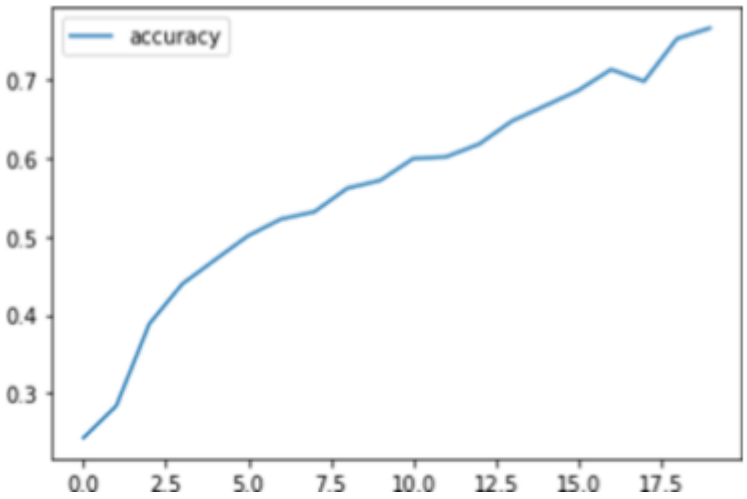
```
model.
↪compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

After compiling the model we then fit the data for training and validation.
Here we are taking batch_size of 512 and epochs =20,we can tune them according to our need.

```
history=model.fit(train_gen,validation_data=test_gen,batch_size=512,epochs = 20)
```

```
Epoch 1/20
57/57 [==============================] - 333s 6s/step - loss: 3.2164 - accuracy:
0.2267 - val_loss: 1.8244 - val_accuracy: 0.2014
Epoch 2/20

57/57 [==============================] - 317s 6s/step - loss: 1.7918 - accuracy:

0.2594 - val_loss: 1.6372 - val_accuracy: 0.3625 Epoch 3/20
57/57 [==============================] - 307s 5s/step - loss: 1.5928 - accuracy:
0.3768 - val_loss: 1.5994 - val_accuracy: 0.3859
Epoch 4/20

57/57 [==============================] - 313s 5s/step - loss: 1.4850 - accuracy:

0.4268 - val_loss: 1.4373 - val_accuracy: 0.4544 Epoch 5/20

57/57 [==============================] - 325s 6s/step - loss: 1.4126 - accuracy:

0.4621 - val_loss: 1.3738 - val_accuracy: 0.4820 Epoch 6/20

57/57 [==============================] - 324s 6s/step - loss: 1.3196 - accuracy:

0.5014 - val_loss: 1.3005 - val_accuracy: 0.5093 Epoch 7/20

57/57 [==============================] - 323s 6s/step - loss: 1.2455 - accuracy:

0.5280 - val_loss: 1.3039 - val_accuracy: 0.4937 Epoch 8/20
57/57 [==============================] - 317s 6s/step - loss: 1.2677 - accuracy:
0.5156 - val_loss: 1.2738 - val_accuracy: 0.5177 Epoch 9/20

57/57 [==============================] - 292s 5s/step - loss: 1.1523 - accuracy:

0.5674 - val_loss: 1.3120 - val_accuracy: 0.5018 Epoch 10/20

57/57 [==============================] - 293s 5s/step - loss: 1.1477 - accuracy:

0.5685 - val_loss: 1.2689 - val_accuracy: 0.5185 Epoch 11/20

57/57 [==============================] - 292s 5s/step - loss: 1.0679 - accuracy:

0.5958 - val_loss: 1.2960 - val_accuracy: 0.5283 Epoch
12/20

57/57 [==============================] - 294s 5s/step - loss: 1.0363 - accuracy:

0.6144 - val_loss: 1.2462 - val_accuracy: 0.5358 Epoch
13/20
57/57 [==============================] - 307s 5s/step - loss: 0.9905 - accuracy:
0.6239 - val_loss: 1.2570 - val_accuracy: 0.5311 Epoch
14/20

57/57 [==============================] - 298s 5s/step - loss: 0.9414 - accuracy:

0.6512 - val_loss: 1.2798 - val_accuracy: 0.5489 Epoch
15/20

57/57 [==============================] - 294s 5s/step - loss: 0.8686 - accuracy:

0.6748 - val_loss: 1.2689 - val_accuracy: 0.5369 Epoch
16/20

57/57 [==============================] - 519s 9s/step - loss: 0.8699 - accuracy:

0.6773 - val_loss: 1.3989 - val_accuracy: 0.5294
```

Here are the codes to print the accuracy plot.



We now serialize the model to JSON and save the model weight in hd5 file
So that we can make use of this file to make predictions rather than training
The network again.Code for this is written below.

```
57/57 [==============================] - 292s 5s/step - loss: 1.0679 - accuracy:

0.5958 - val_loss: 1.2960 - val_accuracy: 0.5283 Epoch
12/20

57/57 [==============================] - 294s 5s/step - loss: 1.0363 - accuracy:

0.6144 - val_loss: 1.2462 - val_accuracy: 0.5358 Epoch
13/20
57/57 [==============================] - 307s 5s/step - loss: 0.9905 - accuracy:
0.6239 - val_loss: 1.2570 - val_accuracy: 0.5311 Epoch
14/20

57/57 [==============================] - 298s 5s/step - loss: 0.9414 - accuracy:

0.6512 - val_loss: 1.2798 - val_accuracy: 0.5489 Epoch
15/20

57/57 [==============================] - 294s 5s/step - loss: 0.8686 - accuracy:

0.6748 - val_loss: 1.2689 - val_accuracy: 0.5369 Epoch
16/20

57/57 [==============================] - 519s 9s/step - loss: 0.8699 - accuracy:

0.6773 - val_loss: 1.3989 - val_accuracy: 0.5294
```

```
fer_json = model.to_json()
with open("fer.json", "w") as json_file: json_file.write(fer_json)
model.save_weights("fer.h5")
```

Testing the model in Real-time using OpenCV and WebCam

Till now we have train our model on our dataset.Now we will test the model that we build for emotion detection in real-time using OpenCV and webcam. To do so we will write a python script. We will use the Jupyter notebook in our local system to make use of a webcam. You can use other IDEs as well. First, we will install a few libraries that are required. Use the below code to import those all.

```
[20]: from keras.models import load_model
from time import sleep
#from keras.models import model_from_jason
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
import cv2
import numpy as np

#model = model_from_json(open("fer.json", "r").read())
#load weights
model.load_weights('fer.h5')
```

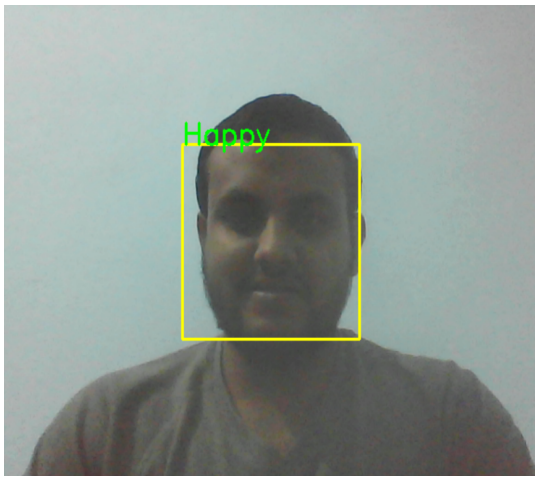
After importing all the required libraries we will load the model weights that we saved earlier after training. Use the below code to load your saved model. After importing the model weights we have imported a haar cascade file that is designed by open cv to detect the frontal face.

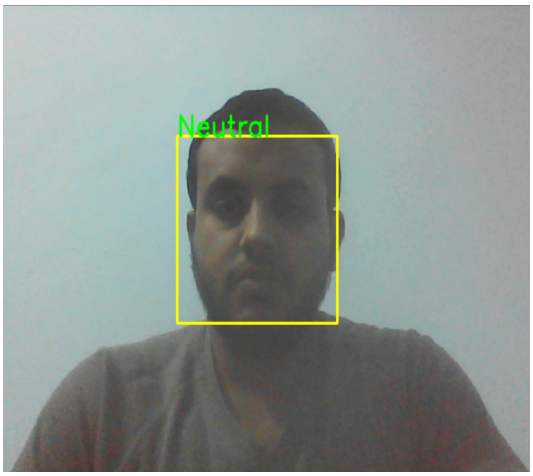
```
face_classifier = cv2.CascadeClassifier(r"C:
↪\Users\HP\Desktop\Emotion-recognition-master\haarcascade_files\haarcascade_frontalface_defaul
↪xml")
```

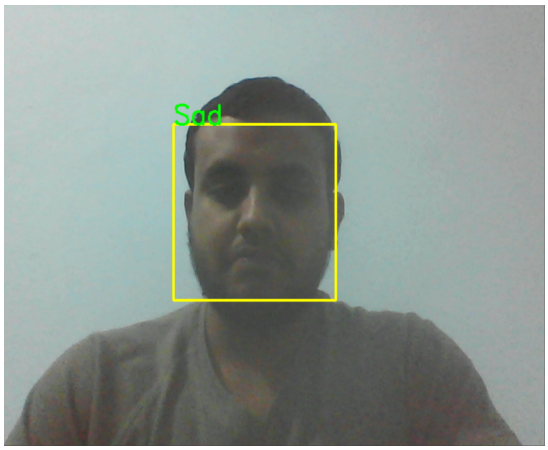
After importing the haar cascade file we will have written a code to detect faces and classify the desired emotions. We have assigned the labels that will be different emotions like angry, happy, sad, surprise, neutral. As soon as you run the code a new window will pop up and your webcam will turn on. It will then detect the face of the person, draw a bounding box over the detected person, and then convert the RGB image into grayscale & classify it in real-time. Please refer to the below code for the same and sample outputs that are shown in the images. To stop the code you need to press ‘q’.

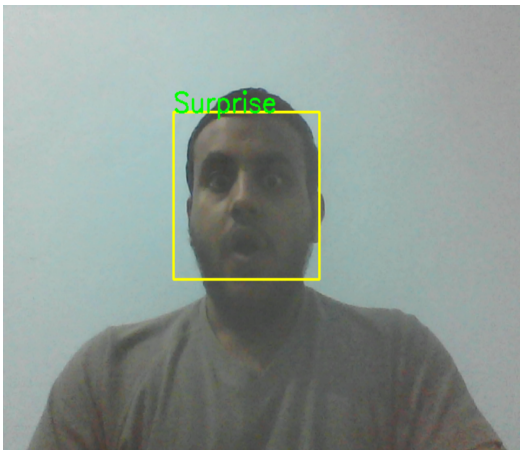
```
emotion_labels = ['Angry','Disgust','Fear','Happy','Sad', 'Surprise', 'Neutral']

cap = cv2.VideoCapture(0)



while True:
    _, frame = cap.read()
    labels = []
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,255),2)
        roi_gray = gray[y:y+h,x:x+w]
        roi_gray = cv2.resize(roi_gray,(48,48),interpolation=cv2.INTER_AREA)
```

```
        if np.sum([roi_gray])!=0:
            roi = roi_gray.astype('float')/255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi,axis=0)

            prediction =model.predict(roi)[0]
#             print(prediction)
            label=emotion_labels[prediction.argmax()]
            label_position = (x,y)
            cv2.putText(frame,label,label_position,cv2.
 ↪FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
        else:
            cv2.putText(frame,'No Faces',(30,80),cv2.
 ↪FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    cv2.imshow('Emotion Detector',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```


Happy


Neutral


Sad


Surprise

Emotion Recognition from Audio

In this mini project,Building and training a simple Speech Emotion Recognizer that predicts human emotions from audio files using Python, Sci-kit learn, librosa, and Keras. Firstly, we will load the data (Ravdess dataset), extract features (MFCC) from it, and split it into training and testing sets. Then, we will initialize CNN model as emotion classifiers and train them. Finally, we will calculate the accuracy of our models.

The whole pipeline is as follows (as same as any machine learning pipeline):

- (1)Loading the Dataset: This process is about loading the dataset in Python which involves extracting audio features, such as MFCC
- (2)Training the Model: After we prepare and load the dataset, we simply train it on a suited model.
- (3)Testing the Model: Measuring how good our model is doing.

```
[2]: import librosa
import matplotlib.pyplot as plt
import librosa.display
from IPython.display import Audio
import numpy as np
import tensorflow as tf
from matplotlib.pyplot import specgram
import pandas as pd
from sklearn.metrics import confusion_matrix
import IPython.display as ipd # To play sound in the notebook import os # interface with underlying OS
that python is running on import sys
import warnings
# ignore warnings
if not sys.warnoptions:
warnings.simplefilter("ignore")
warnings.filterwarnings("ignore", category=DeprecationWarning) from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, AveragePooling1D from
tensorflow.keras.layers import Input, Flatten, Dropout, Activation, ␣ ,→BatchNormalization, Dense
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier from tensorflow.keras.optimizers
import SGD
from tensorflow.keras.regularizers import l2
import seaborn as sns
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint from tensorflow.keras.utils import
to_categorical
from sklearn.metrics import classification_report
```

Dataset:

We have used Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) dataset.

Speech audio-only files (16bit, 48kHz .wav) from the RAVDESS. Full dataset of speech and song, audio and video (24.8 GB) . Construction and perceptual validation of the RAVDESS is described in our Open Access paper in PLoS ONE.

Check out our Kaggle Song emotion dataset.

Files:

This portion of the RAVDESS contains 1440 files: 60 trials per actor x 24 actors = 1440. The RAVDESS contains 24 professional actors (12 female, 12 male), vocalizing two lexically-matched statements in a neutral North American accent. Speech emotions includes calm, happy, sad, angry, fearful, surprise, and disgust expressions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression.

Each of the 1440 files has a unique filename. The filename consists of a 7-part numerical identifier (e.g., 03-01-06-01-02-01-12.wav). These identifiers define the stimulus characteristics.

Filename identifiers**:**

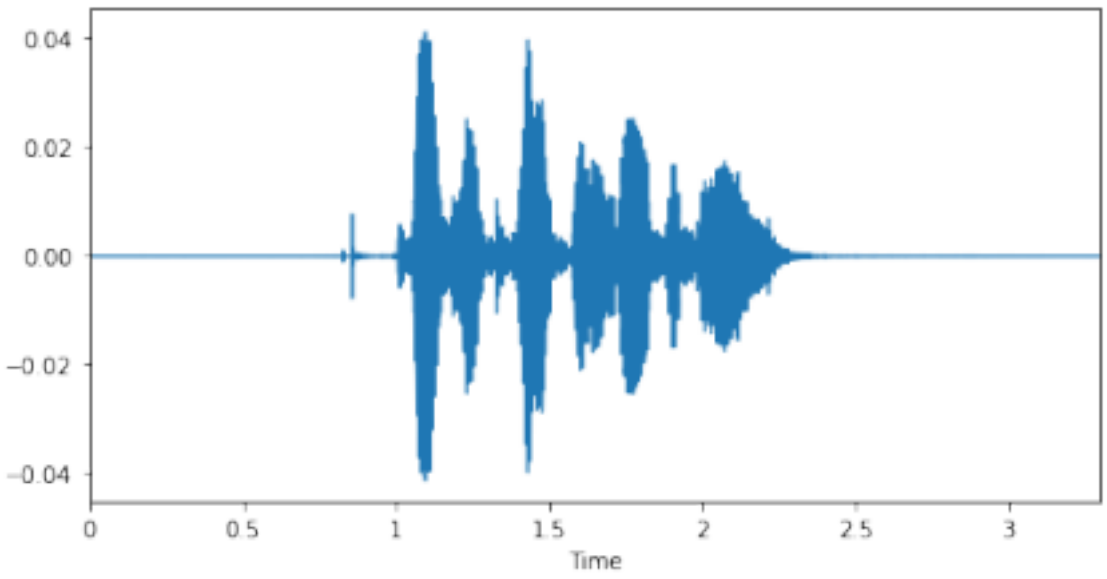
- Modality (01 = full-AV, 02 = video-only, 03 = audio-only).
- Vocal channel (01 = speech, 02 = song).
- Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
- Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
- Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
- Repetition (01 = 1st repetition, 02 = 2nd repetition).
- Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

Librosa :

Librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

```
[3]: x, sr = librosa.load(r'C:\Users\HP\Desktop\dataset␣
↪2\audio_speech_actors_01-24\Actor_01\03-01-01-01-01-01-01.wav')
[4]: plt.figure(figsize=(8, 4))
librosa.display.waveplot(x, sr=sr)
```

[4]: <matplotlib.collections.PolyCollection at 0x27c1511bc70> 3



Mel-frequency Cepstrum Coefficients (MFCC)

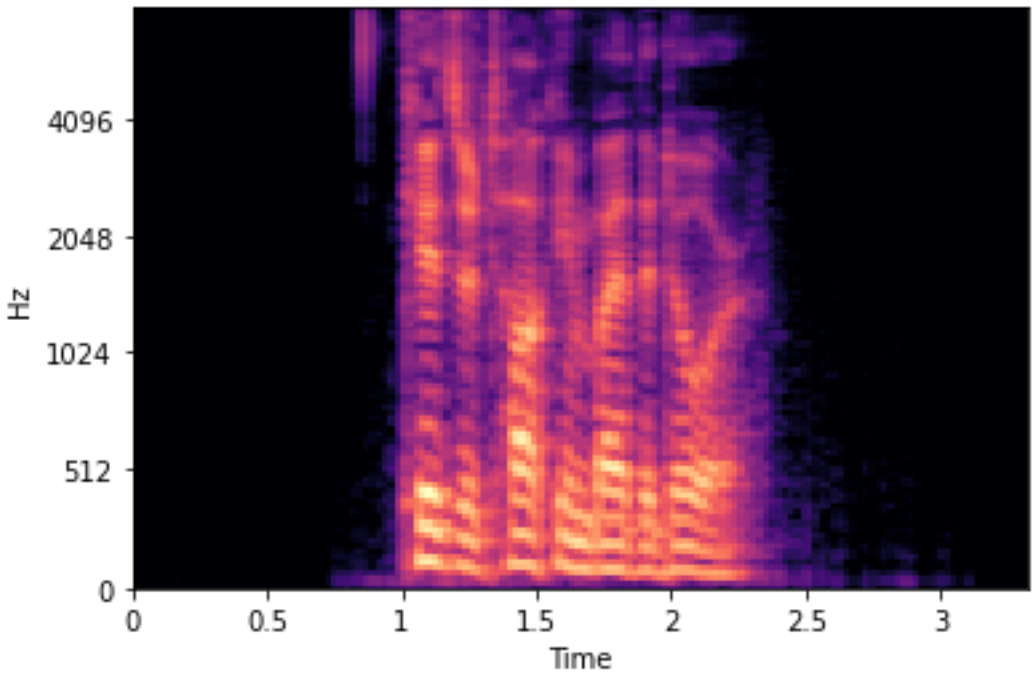
The sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. This shape determines what sound comes out. If we can determine the shape accurately, this should give us an accurate representation of the phoneme being produced. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.

A multilayer perceptron (MLP)

A multilayer perceptron (MLP) is a deep, artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP.

```
[5]: spectrogram = librosa.feature.melspectrogram(y=x, sr=sr, n_mels=128,fmax=8000)
     spectrogram = librosa.power_to_db(spectrogram)
```

```
librosa.display.specshow(spectrogram, y_axis='mel', fmax=8000, x_axis='time');
```



1.Loading the dataset:

```
[6]: audio = r"C:\Users\HP\Desktop\dataset 2\audio_speech_actors_01-24"
     actor_folders = os.listdir(audio) #list files in audio directory actor_folders.sort()
     actor_folders[0:5]
```

[6]: ['Actor_01', 'Actor_02', 'Actor_03', 'Actor_04', 'Actor_05']

```
[7]: emotion = []
     gender = []
     actor = []
     file_path = []
     for i in actor_folders:
         filename = os.listdir(audio + "\\" +i) #iterate over Actor folders for f in filename: # go through
         files in Actor folder
             part = f.split('.')[0].split('-')
             emotion.append(int(part[2]))
             actor.append(int(part[6]))
             bg = int(part[6])
             if bg%2 == 0:
                 bg = "female"
             else:
                 bg = "male"
             gender.append(bg)
             file_path.append(audio + "\\" + i + "\\" + f)
             #print(f)
     #print(file_path)
```

2. Converting our dataset into dataframe

```
[8]: audio_df = pd.DataFrame(emotion)
     audio_df = audio_df.replace({1:'neutral', 2:'calm', 3:'happy', 4:'sad', 5: ↪'angry',
       6:'fear', 7:'disgust', 8:'surprise'})
     audio_df = pd.concat([pd.DataFrame(gender),audio_df,pd.DataFrame(actor)],axis=1)
     audio_df.columns = ['gender','emotion','actor']
     audio_df = pd.concat([audio_df,pd.DataFrame(file_path, columns = ␣ ↪
       ['path'])],axis=1)
     audio_df
```

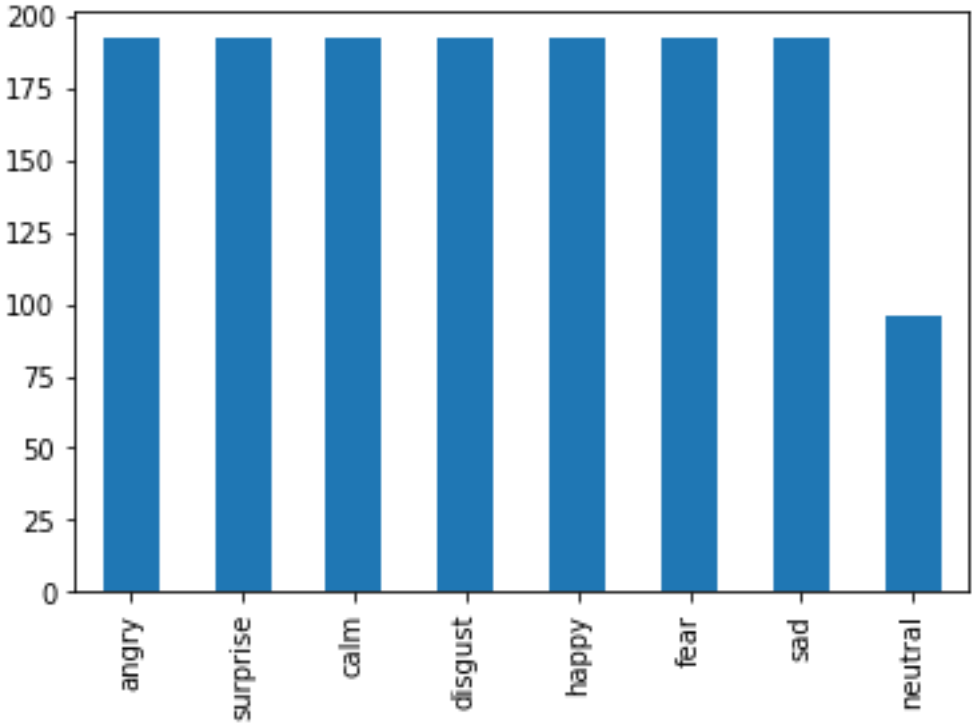
```
[8]: gender emotion actor \
     0 male neutral 1
     1 male neutral 1
     2 male neutral 1
     3 male neutral 1
     4 male calm 1
     ... ... ... ...
     1435 female surprise 24
     1436 female surprise 24
     1437 female surprise 24
     1438 female surprise 24
```



```
     1439 female surprise 24

                                                        path
     0     C:\Users\HP\Desktop\dataset     2\audio_speech_act...     1
     C:\Users\HP\Desktop\dataset         2\audio_speech_act...       2
     C:\Users\HP\Desktop\dataset         2\audio_speech_act...       3
     C:\Users\HP\Desktop\dataset         2\audio_speech_act...       4
     C:\Users\HP\Desktop\dataset  2\audio_speech_act... ... ...  1435
     C:\Users\HP\Desktop\dataset       2\audio_speech_act...      1436
     C:\Users\HP\Desktop\dataset       2\audio_speech_act...      1437
     C:\Users\HP\Desktop\dataset       2\audio_speech_act...      1438
     C:\Users\HP\Desktop\dataset       2\audio_speech_act...      1439
     C:\Users\HP\Desktop\dataset 2\audio_speech_act...

     [1440 rows x 4 columns]
```

```
[10]: audio_df.emotion.value_counts().plot(kind='bar')
```

```
[10]: <AxesSubplot:>
```



[ ]:

```
[11]: audio_df.to_csv('Desktop\\audio.csv')
```

3. Extracting mfcc features

```
[12]: df = pd.DataFrame(columns=['mel_spectrogram'])

      counter=0

    for index,path in enumerate(audio_df.path):
            X, sample_rate = librosa.load(path,␣
       ↪res_type='kaiser_fast',duration=3,sr=44100,offset=0.5)
            #get the mel-scaled spectrogram (ransform both the y-axis (frequency) to log␣ ↪scale, and the "color"
      axis (amplitude) to Decibels, which is kinda the log␣ ↪scale of amplitudes.)
            spectrogram = librosa.feature.melspectrogram(y=X, sr=sample_rate,␣ ↪
      n_mels=128,fmax=8000)
            db_spec = librosa.power_to_db(spectrogram)
            #temporally average spectrogram
            log_spectrogram = np.mean(db_spec, axis = 0)

            # Mel-frequency cepstral coefficients (MFCCs)
      # mfcc = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
      # mfcc=np.mean(mfcc,axis=0)
      # compute chroma energy (pertains to 12 different pitch classes)
      # chroma = librosa.feature.chroma_stft(y=X, sr=sample_rate)
      # chroma = np.mean(chroma, axis = 0)
      # compute spectral contrast
      # contrast = librosa.feature.spectral_contrast(y=X, sr=sample_rate) # contrast =
      np.mean(contrast, axis= 0)
      # compute zero-crossing-rate (zcr:the zcr is the rate of sign changes along␣ ↪a signal i.e.m the
      rate at
      # which the signal changes from positive to negative or back - separation of␣ ↪voiced andunvoiced
        speech.)
      # zcr = librosa.feature.zero_crossing_rate(y=X)
      # zcr = np.mean(zcr, axis= 0)
```

```
[13]: df_combined = pd.concat([audio_df,pd.DataFrame(df['mel_spectrogram'].values. ↪tolist())],axis=1)
      df_combined = df_combined.fillna(0)

[14]: df_combined.drop(columns='path',inplace=True)
```

```
[15]: df_combined.head()
```

[15]: gender emotion actor 0 1 2 3 \ 0 male neutral 1 -76.384773 -76.384773 -76.384773 -76.384773 1 male neutral 1 -75.335518 -75.445320 -75.554031 -75.203949

8

2 male neutral 1 -75.150711 -75.150711 -75.150711 -75.150711 3 male neutral 1 -75.268448 -75.268448 -75.268448 -75.268448 4 male calm 1 -80.147377 -80.147377 -80.147377 -80.147377

4 5 6 ... 249 250 251 \ 0 -76.384773 -76.384773 -76.384773 ... 0.000000 0.000000 0.000000 1 -75.230530 -75.319374 -75.653793 ... 0.000000 0.000000 0.000000 2 -75.150711 -75.150711 -75.150711 ... 0.000000 0.000000 0.000000 3 -75.268448 -75.268448 -75.268448 ... 0.000000 0.000000 0.000000 4 -80.147377 -80.147377 -80.147377 ... -80.147377 -80.147377 -80.121956

252 253 254 255 256 257 258 0 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 2 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 3 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 4 -79.998009 -80.119255 -80.147377 -80.130417 -80.014702 -80.147377 -80.147377

[5 rows x 262 columns]

```
[16]: train,test = train_test_split(df_combined, test_size=0.2, random_state=0, ␣
→stratify=df_combined[['emotion','gender','actor']])
```

4.Converting features into arrays for both training and test data:

```
[17]: X_train = train.iloc[:, 3:]
      y_train = train.iloc[:,:2].drop(columns=['gender'])
      print(X_train.shape)

      (1152, 259)

[18]: X_test = test.iloc[:,3:]
      y_test = test.iloc[:,:2].drop(columns=['gender'])
      print(X_test.shape)

      (288, 259)

[19]: mean = np.mean(X_train, axis=0)
      std = np.std(X_train, axis=0)
      X_train = (X_train - mean)/std
      X_test = (X_test - mean)/std

[20]: X_train = np.array(X_train)
      y_train = np.array(y_train)
      X_test = np.array(X_test)
      y_test = np.array(y_test)
      y_train.shape
```

[20]: (1152, 1)

5. Converting into one hot vector:

```
[21]: lb = LabelEncoder()
      y_train = to_categorical(lb.fit_transform(y_train))
      y_test = to_categorical(lb.fit_transform(y_test))

      print(y_test[0:10])
      print(y_test.shape)
```

```
[[0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]]
(288, 8)
```

```
[22]: print(lb.classes_)

        ['angry' 'calm' 'disgust' 'fear' 'happy' 'neutral' 'sad' 'surprise']

[23]: X_train = X_train[:,:,np.newaxis]
        X_test = X_test[:,:,np.newaxis]

[24]: X_train.shape
```

[24]: (1152, 259, 1)

[25]: X_test.shape

[25]: (288, 259, 1)

6.Neural network:

```
[28]: import tensorflow as tf
        from tensorflow import keras
        from tensorflow.keras import layers
        from tensorflow.keras.models import Sequential, load_model
        from keras.layers import *
        from keras.optimizers import RMSprop

        #model = Sequential()
        # First GRU layer with Dropout regularisation
        #model.add(GRU(units=50, return_sequences=True, input_shape=(X_train.
          ,→shape[1],1), activation='tanh'))
        #model.add(Dropout(0.2))
        # Second GRU layer
        #model.add(GRU(units=50, return_sequences=True, input_shape=(X_train.
          ,→shape[1],1), activation='tanh'))
        #model.add(Dropout(0.2))
        # Third GRU layer
        #model.add(GRU(units=50, return_sequences=True, input_shape=(X_train.
          ,→shape[1],1), activation='tanh'))
        #model.add(Dropout(0.2))
        # Fourth GRU layer
        #model.add(GRU(units=50, activation='tanh'))
        #model.add(Dropout(0.2))
        # The output layer
        #model.add(Dense(units=8))

        #model = Sequential()
        #model.add(LSTM(128, return_sequences=False, input_shape=(40, 1)))
        #model.add(Dense(64))
```

```
#model.add(Dropout(0.4))
#model.add(Activation('relu'))
#model.add(Dense(32))
```



```
#model.add(Dropout(0.4))
#model.add(Activation('relu'))
#model.add(Dense(8))
#model.add(Activation('softmax'))

#BUILD 1D CNN LAYERS
model = tf.keras.Sequential()
model.add(layers.Conv1D(64, kernel_size=(10), activation='relu',␣
  ↪input_shape=(X_train.shape[1],1)))
model.add(layers.Conv1D(128,␣
  ↪kernel_size=(10),activation='relu',kernel_regularizer=l2(0.01),␣
  ↪bias_regularizer=l2(0.01)))
model.add(layers.MaxPooling1D(pool_size=(8)))
model.add(layers.Dropout(0.4))
model.add(layers.Conv1D(128, kernel_size=(10),activation='relu'))
model.add(layers.MaxPooling1D(pool_size=(8)))
model.add(layers.Dropout(0.4))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(8, activation='sigmoid'))
opt = keras.optimizers.Adam(lr=0.001)
model.
  ↪compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

Model: "sequential"

_____ Layer (type) Output Shape Param #
========== conv1d (Conv1D) (None, 250, 64) 704
_____ conv1d_1 (Conv1D) (None, 241, 128) 82048
_____ max_pooling1d (MaxPooling1D) (None, 30, 128) 0
_____ dropout (Dropout) (None, 30, 128) 0
_____ conv1d_2 (Conv1D) (None, 21, 128) 163968
_____ max_pooling1d_1 (MaxPooling1 (None, 2, 128) 0

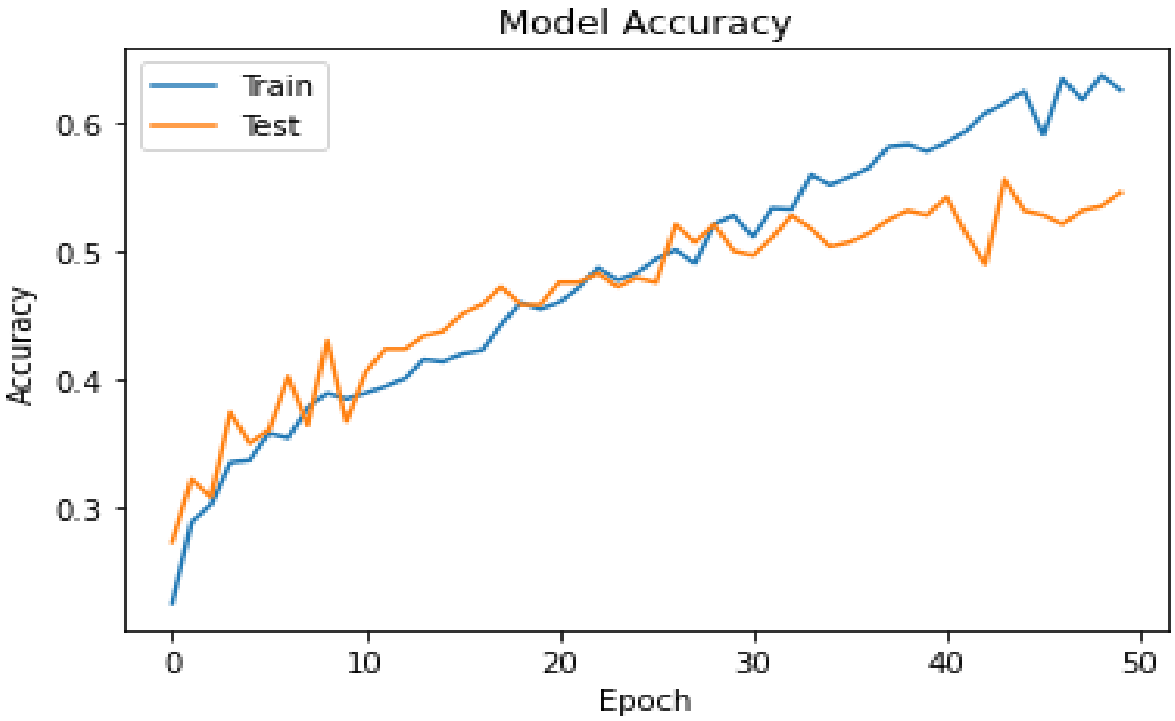
_____ dropout_1 (Dropout) (None, 2, 128) 0
_____
_____ flatten (Flatten) (None, 256) 0
_____
_____ dense (Dense) (None, 256) 65792

_____
_____ dropout_2 (Dropout) (None, 256) 0
_____
_____ dense_1 (Dense) (None, 8) 2056
==========
======== Total params: 314,568
Trainable params: 314,568
Non-trainable params: 0
_____

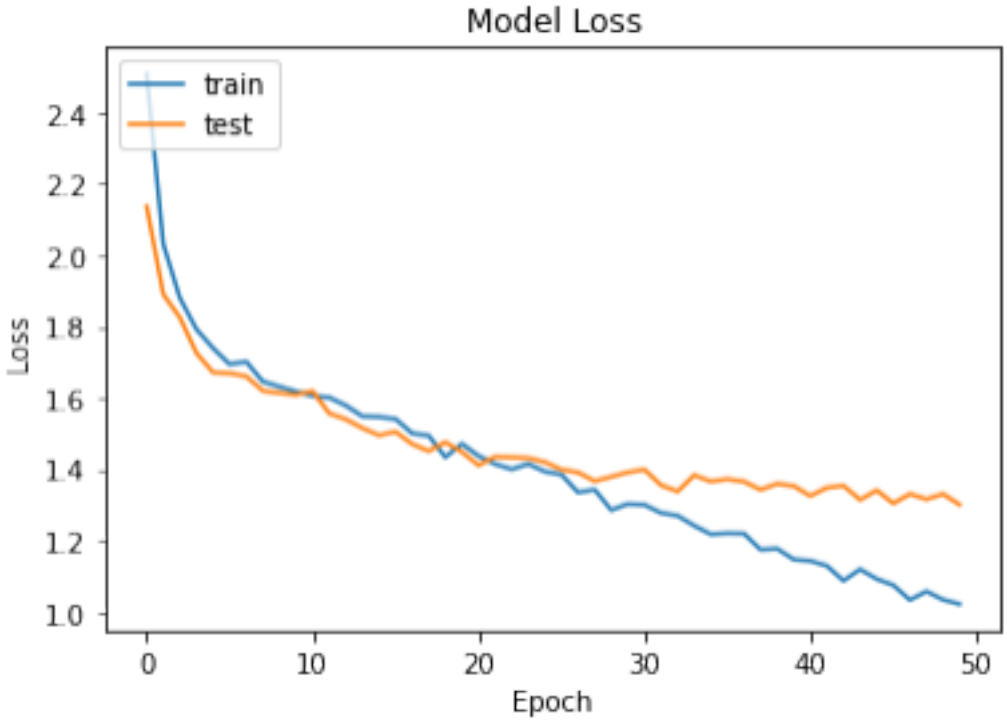
```
[29]: import tensorflow.keras as keras

# FIT MODEL AND USE CHECKPOINT TO SAVE BEST MODEL
checkpoint = ModelCheckpoint("best_initial_model.hdf5", monitor='val_accuracy', verbose=1,
    save_best_only=True, mode='max', period=1, save_weights_only=True)

model_history=model.fit(X_train, y_train,batch_size=32, epochs=50,
  validation_data=(X_test, y_test),callbacks=[checkpoint])
```

Now we have done model fitting with batchsize=32 and epochs=50 and saved the model. After 50 epochs we got accuracy as 62.53% and validation accuracy as 54.51%.

```
[30]: # PLOT MODEL HISTORY OF ACCURACY AND LOSS OVER EPOCHS
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.savefig('Initial_Model_Accuracy.png')
plt.show()
# summarize history for loss
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.savefig('Initial_Model_loss.png')
plt.show()
```


Model Accuracy
Train
Test
Accuracy
0.6
0.5
0.4
0.3
0
10
20
30
40
50
Epoch

20


Model Loss
train
test
Loss
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0
10
20
30
40
50
Epoch

```
[33]: # PREDICTIONS
      predictions = model.predict(X_test)
      predictions=predictions.argmax(axis=1)
      predictions = predictions.astype(int).flatten()
      predictions = (lb.inverse_transform((predictions)))
      predictions = pd.DataFrame({'Predicted Values': predictions})

      # ACTUAL LABELS
      actual=y_test.argmax(axis=1)
      actual = actual.astype(int).flatten()
      actual = (lb.inverse_transform((actual)))
      actual = pd.DataFrame({'Actual Values': actual})

      # COMBINE BOTH
```

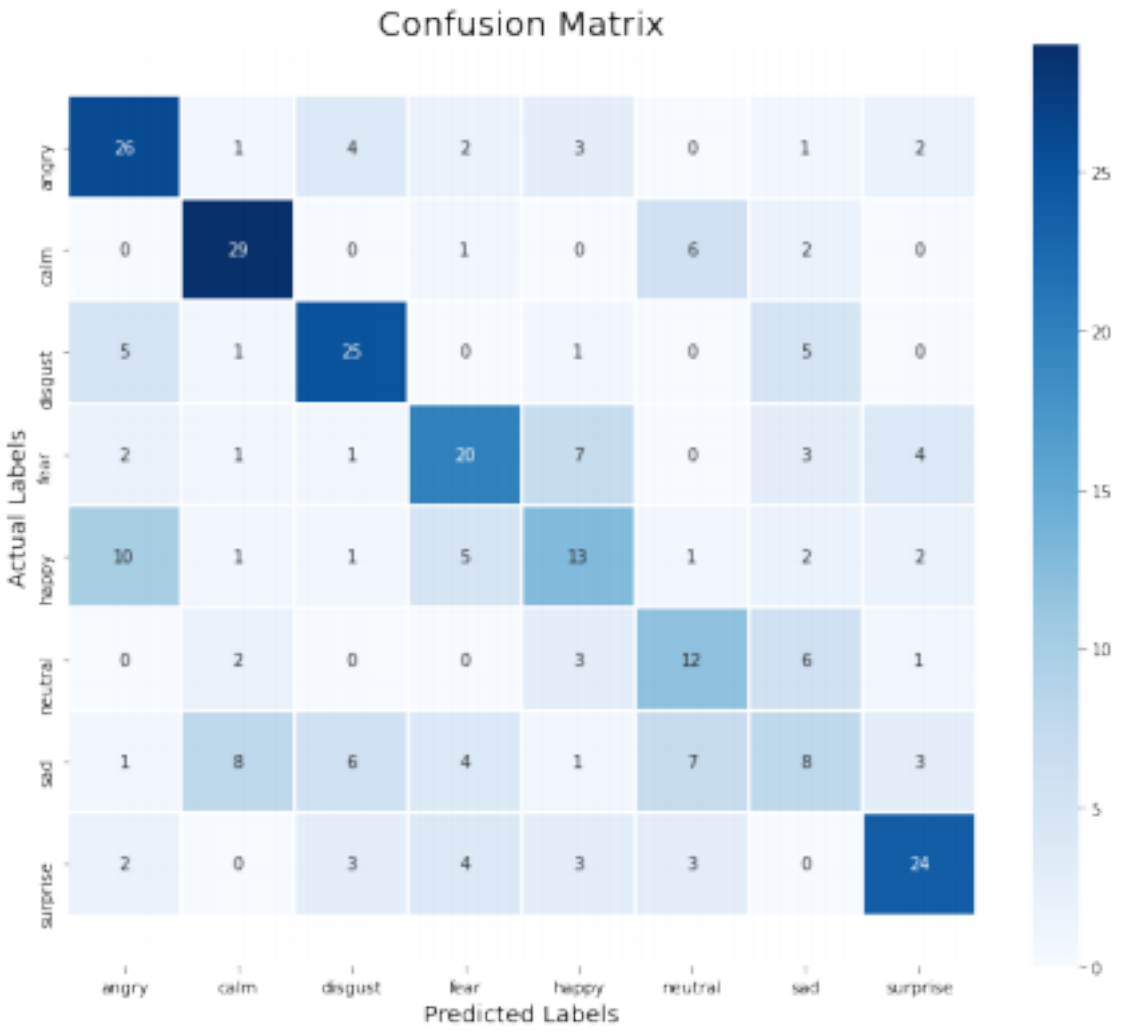


```
      finaldf = actual.join(predictions)
      finaldf[140:150]
```

[33]: Actual Values Predicted Values
140 sad neutral
141 surprise happy
142 neutral sad
143 sad sad
144 fear fear
145 sad sad
146 disgust disgust
147 angry angry
148 surprise happy
149 angry angr

7.Print Confusion Matrix:

```
[34]: # CREATE CONFUSION MATRIX OF ACTUAL VS. PREDICTION
      cm = confusion_matrix(actual, predictions)
      plt.figure(figsize = (12, 10))
      cm = pd.DataFrame(cm , index = [i for i in lb.classes_] , columns = [i for i in␣ ,
        →lb.classes_])
      ax = sns.heatmap(cm, linecolor='white', cmap='Blues', linewidth=1, annot=True,␣ ,
        →fmt='')
      bottom, top = ax.get_ylim()
      ax.set_ylim(bottom + 0.5, top - 0.5)
      plt.title('Confusion Matrix', size=20)
      plt.xlabel('Predicted Labels', size=14)
      plt.ylabel('Actual Labels', size=14)
      plt.savefig('Initial_Model_Confusion_Matrix.png')
      plt.show()
```



8.Print classification report:

```
[35]: print(classification_report(actual, predictions, target_names =␣ ↪
          ['angry','calm','disgust','fear','happy','neutral','sad','surprise']))

                        precision recall f1-score support

                 angry 0.57 0.67 0.61 39
                  calm 0.67 0.76 0.72 38
               disgust 0.62 0.68 0.65 37
                  fear 0.56 0.53 0.54 38
                 happy 0.42 0.37 0.39 35
               neutral 0.41 0.50 0.45 24
                   sad 0.30 0.21 0.25 38
              surprise 0.67 0.62 0.64 39

              accuracy 0.55 288
             macro avg 0.53 0.54 0.53 288
```



```
          weighted avg 0.53 0.55 0.54 288
```