Empirical Evaluation of speech emotion recognition

1st Suhas Dhar, 2nd Deep Patel

***Abstract*—The emotional state of the speaker has an impact on human speech, which is produced by the vocal cord vibrating. Accurately identifying the various emotions cloaked in human speech will help to significantly increase the caliber of human-computer interaction (HCI). However, the lack of a widely acknowledged standard feature set is the fundamental reason why an acceptable level of accuracy has not yet been attained. Even humans find it challenging to discriminate between speech and emotions, which is why it is challenging to extract the standard feature set. In comparison to the current state of the art, this research proposes a model to accurately classify emotions from speech data. Speech recordings specifically tagged with various emotions were included in the speech dataset utilized in this study. The emotional state of the speaker has an impact on human speech, which is produced by the vocal cord vibrating. Accurately identifying the various emotions cloaked in human speech will help to significantly increase the caliber of human-computer interaction (HCI). However, the lack of a widely acknowledged standard feature set is the fundamental reason why an acceptable level of accuracy has not yet been attained. Even humans find it challenging to discriminate between speech and emotions, which is why it is challenging to extract the standard feature set. In comparison to the current state of the art, this research proposes a model to accurately classify emotions from speech data. Speech recordings specifically tagged with various emotions were included in the speech dataset utilized in this study.**

***Index Terms*—Multi-Layer Perceptron, Machine Learning, Emotion Recognition, Data Augmentation**

I. Introduction

The project is aimed at reviewing the effectiveness of using the machine learning model CNN on a classification problem of recognizing the emotion of a speech. data in this case would consist of a combination of words in English and unreliable sources would be picked up for the base data.

Upon receipt of data, if any sort of cleansing is required, we would use the best industry practices to perform the cleansing before the data is ready to be processed by the machine learning model. this would be followed by an empirical analysis of what machine learning models would best suit the data like this post which we will train the model using a part of the data set. The other part of this data set would be used as the testing part.

The problem would be a classification problem where we would have four to five classes. At this point, we are yet to evaluate if the input would be in form of sound signals or as the text-speech. if we conclude to have the sound input then the input needs to be converted into a waveform to get the attributes of the data. In case the input is in the form of text we can have the attributes identified straightforwardly.

II. Motivation:

There have been many attempts to understand the mood of the person by classifying the features of his speech. the motivation behind this project was to convert the audio files into some interpret able data which could further be used for getting the emotion the person is carrying while he's delivering the speech. Memory and social perceptions are influenced by one's mood. Speech is the quickest and most natural way for humans to communicate. This reality has prompted many academics to examine voice signals as a rapid and effective way for computers and humans to connect. It implies that the computer should be able to recognize human voices and speech. Although there has been substantial progress in voice recognition, researchers are still avoiding the natural interplay between computers and humans since computers are incapable of comprehending human emotional states.

III. Main Contributions and Objectives:

- Data Preprocessing
- Divide test and train data
- Normalization of the Data Set.
- Review the CNN model
- Build the Convolutional Neural Network
- Compile the data on the model.
- Model fitting
- Confusion matrix generation

A convolutional neural network will be used (CNN). Due to its feature extraction and classification components, CNNs often make effective classifiers and do particularly well with image classification tasks. Like they are effective at discovering patterns within photos.

We'll employ a sequential model, starting with a straightforward model architecture made up of four Conv2D convolution layers, with a dense layer serving as the final output layer. The number of possible classifications is matched by the number of nodes in our output layer, which is 10 (num labels). The model will be trained. We will begin with a small batch size and low number of epochs because training a CNN can take a long time.

The baseline architect of the neural network would be the convolutional neural network. we are yet to take a call on the number of convolutional layers and connected layers. we are anticipating an arrangement of convolutional layers with Max pooling or average pooling layers.

A. *Expected Outcome:*

We anticipate the following results on running various experiments using different databases: The accuracy of the test data may vary across different architectures. We expect to have a confusion matrix that would further help us evaluate the characteristics of the outcomes such as efficiency, effectiveness, accuracy, etc.

IV. Related Work:

A. Datasets used in this project :

- Crowd-sourced Emotional Multimodal Actors Dataset (Crema-D)
- Ryerson Audio-Visual Database of Emotional Speech and Song (Ravdess)
- Surrey Audio-Visual Expressed Emotion (Savee)
- Toronto emotional speech set (Tess)
- librosa is a Python library for analyzing audio and music. It can be used to extract the data from the audio files

```
In [3]: mport pandas as pd
mport numpy as np

mport os
mport sys

 librosa is a Python library for analyzing audio and music. It can be used to extract the data from the audio files
 we will see it later.
mport librosa
mport librosa.display
mport seaborn as sns
mport matplotlib.pyplot as plt

rom sklearn.preprocessing import StandardScaler, OneHotEncoder
rom sklearn.metrics import confusion_matrix, classification_report
rom sklearn.model_selection import train_test_split

 # to play the audio files
 from IPython.display import Audio

 import tensorflow as tf
 from tensorflow import keras
 from tensorflow.keras.callbacks import ReduceLROnPlateau
 from tensorflow.keras.models import Sequential
 from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization
 from tensorflow.keras.utils import to_categorical
 from tensorflow.keras.callbacks import ModelCheckpoint

mport warnings
f not sys.warnoptions:
   warnings.simplefilter("ignore")
arnings.filterwarnings("ignore", category=DeprecationWarning)
```

Here are the filename identifiers as per the official RAVDESS website as planned to be used in the project:

- 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).

```
In [5]: ravdess_directory_list = os.listdir(Ravdess)

file_emotion = []
file_path = []
for dir in ravdess_directory_list:
    # as their are 20 different actors in our previous directory we need to extract files for each actor.
    actor = os.listdir(Ravdess + dir)
    for file in actor:
        part = file.split('.')[0]
        part = part.split('-')
        # third part in each file represents the emotion associated to that file.
        file_emotion.append(int(part[2]))
        file_path.append(Ravdess + dir + '/' + file)

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Ravdess_df = pd.concat([emotion_df, path_df], axis=1)

# changing integers to actual emotions.
Ravdess_df.Emotions.replace(
    {1: 'neutral', 2: 'calm', 3: 'happy', 4: 'sad', 5: 'angry', 6: 'fear', 7: 'disgust', 8: 'surprise'}, inplace=True
Ravdess_df.head()
```

```
In [19]: crema_directory_list = os.listdir(Crema)

file_emotion = []
file_path = []

for file in crema_directory_list:
    # storing file paths
    file_path.append(Crema + file)
    # storing file emotions
    part = file.split('_')
    if part[2] == 'SAD':
        file_emotion.append('sad')
    elif part[2] == 'ANG':
        file_emotion.append('angry')
    elif part[2] == 'DIS':
        file_emotion.append('disgust')
    elif part[2] == 'FEA':
        file_emotion.append('fear')
    elif part[2] == 'HAP':
        file_emotion.append('happy')
    elif part[2] == 'NEU':
        file_emotion.append('neutral')
    else:
        file_emotion.append('Unknown')

# dataframe for emotion of files

emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Crema_df = pd.concat([emotion_df, path_df], axis=1)
Crema_df.head()
```

B. Detail design of Features

- There are 20 different actors in our previous directory we need to extract files for each actor.
- As per structure, the third part in each file represents the emotion associated to that file.
- Create a dictionary of integers mapped to actual emotions.

The audio files in this dataset are named in such a way that the prefix letters describe the emotion classes as follows:

- 'a' = 'anger'
- 'd' = 'disgust'
- 'f' = 'fear'
- 'h' = 'happiness'
- 'n' = 'neutral'
- 'sa' = 'sadness'
- 'su' = 'surprise

C. Data Description

5 | TESS Dataset

```
In [ ]: tess_directory_list = os.listdir(Tess)

file_emotion = []
file_path = []

for dir in tess_directory_list:
    directories = os.listdir(Tess + dir)
    for file in directories:
        part = file.split('.')[0]
        part = part.split('_')[2]
        if part=='ps':
            file_emotion.append('surprise')
        else:
            file_emotion.append(part)
        file_path.append(Tess + dir + '/' + file)

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Tess_df = pd.concat([emotion_df, path_df], axis=1)
Tess_df.head()
```

```
In [ ]: savee_directory_list = os.listdir(Savee)

file_emotion = []
file_path = []

for file in savee_directory_list:
    file_path.append(Savee + file)
    part = file.split('_')[1]
    ele = part[:-6]
    if ele == 'a':
        file_emotion.append('angry')
    elif ele == 'd':
        file_emotion.append('disgust')
    elif ele == 'f':
        file_emotion.append('fear')
    elif ele == 'h':
        file_emotion.append('happy')
    elif ele == 'n':
        file_emotion.append('neutral')
    elif ele == 'sa':
        file_emotion.append('sad')
    else:
        file_emotion.append('surprise')

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Savee_df = pd.concat([emotion_df, path_df], axis=1)
Savee_df.head()
```

D. Data Visualization

We plan to use the below 2 techniques to visualize the data frames formed so far.

- Waveplots - Waveplots let us know the loudness of the audio at a given time.
- Spectograms - A spectrogram is a visual representation of the spectrum of frequencies of sound or other signals as they vary with time. It's a representation of frequencies changing with respect to time for given audio/music signals.

V. Proposed Framework

```
In [ ]: def create_waveplot(data, sr, e):
    plt.figure(figsize=(14, 4))
    plt.title('Waveplot for audio with {} emotion'.format(e), size=15)
    librosa.display.waveshow(data, sr=sr,color='#CC6C6C')
    plt.show()


def create_spectrogram(data, sr, e):
    # stft function converts the data into short term fourier transform
    X = librosa.stft(data)
    Xdb = librosa.amplitude_to_db(abs(X))
    plt.figure(figsize=(14, 4))
    plt.title('Spectrogram for audio with {} emotion'.format(e), size=15)
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
    # librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
    plt.colorbar()
```

7.1 | Fear Emotion

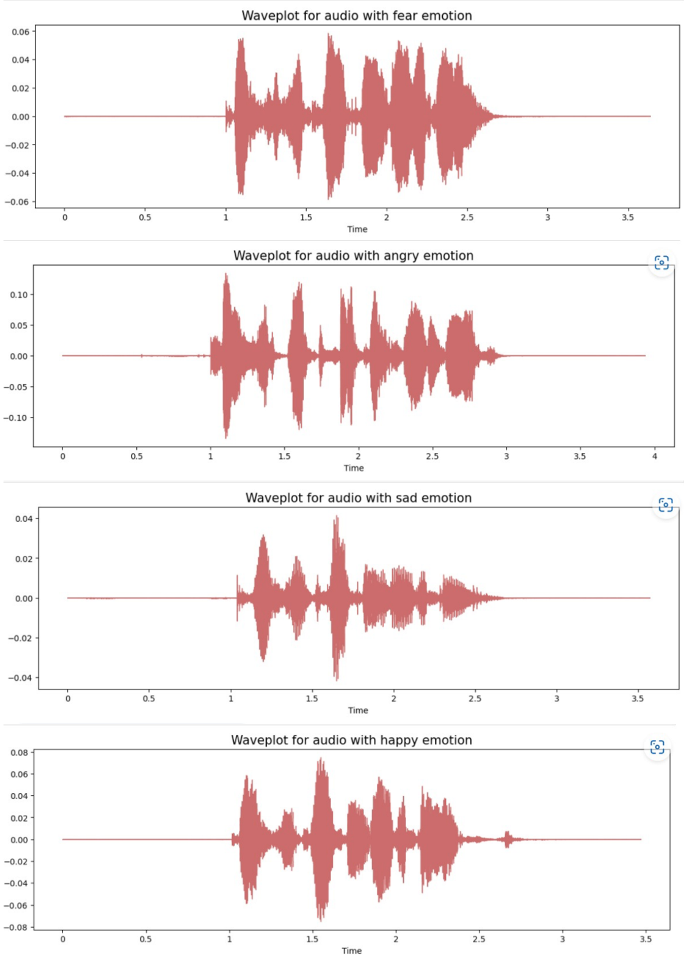
```
In [ ]: emotion = 'fear'
path = np.array(data_path.Path[data_path.Emotions == emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

7.2 | Angry Emotion

```
In [ ]: emotion = 'angry'
path = np.array(data_path.Path[data_path.Emotions == emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

7.4 | Happy Emotion

```
In [ ]: emotion = 'happy'
path = np.array(data_path.Path[data_path.Emotions == emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



A. Data Augmentation

- Data augmentation is the process by which we create new synthetic data samples by adding small perturbations on our initial training set.
- To generate syntactic data for audio, we can apply noise injection, shifting time, changing pitch and speed.
- The objective is to make our model invariant to those perturbations and enhace its ability to generalize.
- In order to this to work adding the perturbations must conserve the same label as the original training sample.
- In images data augmentation can be performed by shifting the image, zooming, rotating.

```
In [ ]: def noise(data):
    noise_amp = 0.035 * np.random.uniform() * np.amax(data)
    data = data + noise_amp * np.random.normal(size=data.shape[0])
    return data


def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate)


def shift(data):
    shift_range = int(np.random.uniform(low=-5, high=5) * 1000)
    return np.roll(data, shift_range)


def pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sampling_rate, pitch_factor)

# taking any example and checking for techniques.
path = np.array(data_path.Path)[1]
data, sample_rate = librosa.load(path)
```

B. Feature Extraction

Extraction of features is a very important part of analyzing and finding relations between different things. As we already know that the data provided by audio cannot be understood by the models directly, so we need to convert them into an understandable format for which feature extraction is used. The audio signal is a three-dimensional signal in which three axes represent time, amplitude, and frequency. As stated there with the help of the sample rate and the sample data, one can perform several transformations on it to extract valuable features.

- Zero Crossing Rate : The rate of sign-changes of the signal during the duration of a particular frame.
- Energy : The sum of squares of the signal values, normalized by the respective frame length.
- Entropy of Energy : The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
- Spectral Centroid : The center of gravity of the spectrum.
- Spectral Spread : The second central moment of the spectrum.
- Spectral Entropy : Entropy of the normalized spectral energies for a set of sub-frames.
- Spectral Flux : The squared difference between the normalized magnitudes of the spectra of the two successive frames.
- Spectral Roll off : The frequency below which 90 percent of the magnitude distribution of the spectrum is concentrated.
- MFCCs Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.
- Chroma Vector : A 12-element representation of the spectral energy where the bins represent the
- equal-tempered pitch classes of western-type music (semitone spacing).
- Chroma Deviation : The standard deviation of the 12 chroma coefficients.

In this project i am not going deep in feature selection process to check which features are good for our dataset rather i am only extracting 5 features:

- Zero Crossing Rate
- Chroma stft
- MFCC
- RMS(root mean square) value
- MelSpectogram to train our model.

So far, the work has been done on data extraction and feature formation. The upcoming reports will have the implementation of Machine Learning models to this data.

```
In [ ]: def extract_features(data):
    # ZCR
    result = np.array([])
    zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
    result=np.hstack((result, zcr)) # stacking horizontally

    # Chroma_stft
    stft = np.abs(librosa.stft(data))
    chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
    result = np.hstack((result, chroma_stft)) # stacking horizontally

    # MFCC
    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mfcc)) # stacking horizontally

    # Root Mean Square Value
    rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
    result = np.hstack((result, rms)) # stacking horizontally

    # MelSpectogram
    mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mel)) # stacking horizontally

    return result
```

```
def get_features(path):
    # duration and offset are used to take care of the no audio in start and the ending of each audio files as seen a
    data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)

    # without augmentation
    res1 = extract_features(data)
    result = np.array(res1)

    # data with noise
    noise_data = noise(data)
    res2 = extract_features(noise_data)
    result = np.vstack((result, res2)) # stacking vertically

    # data with stretching and pitching
    new_data = stretch(data)
    data_stretch_pitch = pitch(new_data, sample_rate)
    res3 = extract_features(data_stretch_pitch)
    result = np.vstack((result, res3)) # stacking vertically

    return result
```

```
In [ ]: len(X), len(Y), data_path.Path.shape

In [ ]: Features = pd.DataFrame(X)
        Features['labels'] = Y
        Features.to_csv('features.csv', index=False)
        Features.head()

In [ ]: X = Features.iloc[: ,:-1].values
        Y = Features['labels'].values

In [ ]: # As this is a multiclass classification problem onehotencoding our Y.
        encoder = OneHotEncoder()
        Y = encoder.fit_transform(np.array(Y).reshape(-1,1)).toarray()

In [ ]: # splitting data
        x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=0, shuffle=True)
        x_train.shape, y_train.shape, x_test.shape, y_test.shape

In [ ]: # scaling our data with sklearn's Standard scaler
        scaler = StandardScaler()
        x_train = scaler.fit_transform(x_train)
        x_test = scaler.transform(x_test)
        x_train.shape, y_train.shape, x_test.shape, y_test.shape

In [ ]: # making our data compatible to model.
        x_train = np.expand_dims(x_train, axis=2)
        x_test = np.expand_dims(x_test, axis=2)
        x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

C. Adding to model

A Sequential model is appropriate for a simple stack of layers with exactly one input and one output tensor. You can also incrementally build a Sequential model using the add() method. 'relu' is a non-linear activation function that is used in multi-layer neural networks

```
[28]: model=Sequential()
      model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu', input_shape=(x_train.shape[1], 1)))
      model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

      model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu'))
      model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

      model.add(Conv1D(128, kernel_size=5, strides=1, padding='same', activation='relu'))
      model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))
      model.add(Dropout(0.2))

      model.add(Conv1D(64, kernel_size=5, strides=1, padding='same', activation='relu'))
      model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

      model.add(Flatten())
      model.add(Dense(units=32, activation='relu'))
      model.add(Dropout(0.3))

      model.add(Dense(units=8, activation='softmax'))
      model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])

      model.summary()
```

VI. Results/ Experimentation and Comparison/Analysis

Summary of the model:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 162, 256)          1536

 max_pooling1d (MaxPooling1D  (None, 81, 256)          0
 )

 conv1d_1 (Conv1D)           (None, 81, 256)           327936

 max_pooling1d_1 (MaxPooling  (None, 41, 256)          0
 1D)

 conv1d_2 (Conv1D)           (None, 41, 128)           163968

 max_pooling1d_2 (MaxPooling  (None, 21, 128)          0
 1D)

 dropout (Dropout)           (None, 21, 128)           0

 conv1d_3 (Conv1D)           (None, 21, 64)            41024

 max_pooling1d_3 (MaxPooling  (None, 11, 64)           0
 1D)

 flatten (Flatten)           (None, 704)               0

 dense (Dense)               (None, 32)                22560

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 8)                 264

=================================================================
Total params: 557,288
Trainable params: 557,288
Non-trainable params: 0
_________________________________________________________________
```

When a statistic has ceased improving, reduce the learning rate. When learning becomes stagnant, models frequently benefit from slowing the learning rate by a factor of 2-10. This callback watches a quantity and reduces the learning rate if no progress is noticed after a 'patience' number of epochs.

Model.fit() trains the data based on parameters given below.

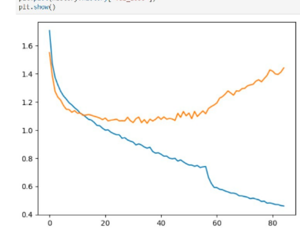
A. We reached an accuracy of 61.9 percent.

```
[29]: rlrp = ReduceLROnPlateau(monitor='loss', factor=0.4, verbose=1, patience=2, min_lr=0.0000001)
      history=model.fit(x_train, y_train, batch_size=64, epochs=85, validation_data=(x_test, y_test), callbacks=[rlrp])

Epoch 1/85
428/428 [==============================] - 96s 221ms/step - loss: 1.7062 - accuracy: 0.3059 - val_loss: 1.5491 - val_accuracy: 0.3713 - lr: 0.0010
Epoch 2/85
428/428 [==============================] - 90s 210ms/step - loss: 1.4703 - accuracy: 0.4088 - val_loss: 1.3866 - val_accuracy: 0.4465 - lr: 0.0010
Epoch 3/85
428/428 [==============================] - 92s 214ms/step - loss: 1.3732 - accuracy: 0.4471 - val_loss: 1.2735 - val_accuracy: 0.4888 - lr: 0.0010
Epoch 4/85
428/428 [==============================] - 91s 213ms/step - loss: 1.3198 - accuracy: 0.4666 - val_loss: 1.2335 - val_accuracy: 0.4943 - lr: 0.0010
Epoch 5/85
428/428 [==============================] - 90s 210ms/step - loss: 1.2750 - accuracy: 0.4870 - val_loss: 1.2121 - val_accuracy: 0.4986 - lr: 0.0010
```

```
Epoch 75/85
428/428 [==============================] - 100s 234ms/step - loss: 0.5094 - accuracy: 0.8037 - val_loss: 1.3468 - val_accuracy: 0.6187 - lr: 4.0000e-04
Epoch 76/85
428/428 [==============================] - 91s 213ms/step - loss: 0.5039 - accuracy: 0.8042 - val_loss: 1.3573 - val_accuracy: 0.6160 - lr: 4.0000e-04
Epoch 77/85
428/428 [==============================] - 144s 338ms/step - loss: 0.4940 - accuracy: 0.8073 - val_loss: 1.3654 - val_accuracy: 0.6183 - lr: 4.0000e-04
Epoch 79/85
428/428 [==============================] - 112s 263ms/step - loss: 0.4800 - accuracy: 0.8138 - val_loss: 1.3873 - val_accuracy: 0.6184 - lr: 4.0000e-04
Epoch 80/85
428/428 [==============================] - 98s 230ms/step - loss: 0.4813 - accuracy: 0.8148 - val_loss: 1.4268 - val_accuracy: 0.6170 - lr: 4.0000e-04
Epoch 81/85
428/428 [==============================] - 95s 222ms/step - loss: 0.4771 - accuracy: 0.8178 - val_loss: 1.4179 - val_accuracy: 0.6148 - lr: 4.0000e-04
Epoch 82/85
428/428 [==============================] - 92s 215ms/step - loss: 0.4706 - accuracy: 0.8206 - val_loss: 1.3979 - val_accuracy: 0.6177 - lr: 4.0000e-04
Epoch 83/85
428/428 [==============================] - 97s 228ms/step - loss: 0.4696 - accuracy: 0.8196 - val_loss: 1.3953 - val_accuracy: 0.6215 - lr: 4.0000e-04
Epoch 84/85
428/428 [==============================] - 95s 223ms/step - loss: 0.4631 - accuracy: 0.8218 - val_loss: 1.4118 - val_accuracy: 0.6195 - lr: 4.0000e-04
Epoch 85/85
428/428 [==============================] - 94s 221ms/step - loss: 0.4597 - accuracy: 0.8244 - val_loss: 1.4410 - val_accuracy: 0.6165 - lr: 4.0000e-04
```

B. The below plot shows the history loss:

```
[32]: plt.plot(history.history["loss"])
      plt.plot(history.history["val_loss"])
      plt.show()
```



C. Prediction on test data

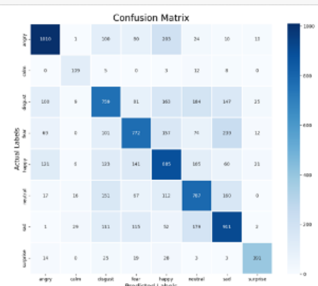
```
In [34]: # predicting on test data.
         pred_test = model.predict(x_test)
         y_pred = encoder.inverse_transform(pred_test)

         y_test = encoder.inverse_transform(y_test)

286/286 [==============================] - 10s 36ms/step

In [35]: df = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
         df['Predicted Labels'] = y_pred.flatten()
         df['Actual Labels'] = y_test.flatten()
         df.head(10)
```

Out[35]:

	Predicted Labels	Actual Labels
0	sad	disgust
1	disgust	disgust
2	angry	angry
3	disgust	disgust
4	fear	fear
5	disgust	fear
6	disgust	happy
7	happy	happy
8	disgust	sad
9	neutral	sad

Generating the Confusion Matrix:

```
In [36]: cm = confusion_matrix(y_test, y_pred)
         plt.figure(figsize = (12, 10))
         cm = pd.DataFrame(cm , index = [i for i in encoder.categories_] , columns = [i for i in encoder.categories_])
         sns.heatmap(cm, linecolor='white', cmap='Blues', linewidth=1, annot=True, fmt='')
         plt.title('Confusion Matrix', size=20)
         plt.xlabel('Predicted Labels', size=14)
         plt.ylabel('Actual Labels', size=14)
         plt.show()
```



D. Below is the classification of the data based on the important features.

```
In [37]: print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

       angry       0.76      0.70      0.73      1438
        calm       0.64      0.80      0.71       137
     disgust       0.55      0.52      0.53      1468
        fear       0.61      0.54      0.57      1424
       happy       0.55      0.61      0.58      1462
     neutral       0.58      0.60      0.59      1310
         sad       0.59      0.65      0.62      1400
    surprise       0.85      0.81      0.83       483

    accuracy                           0.62      9122
   macro avg       0.64      0.65      0.64      9122
weighted avg       0.62      0.62      0.62      9122
```

VII. Responsibility (Task, Person):

Suhas Dhar	Deep Patel
Initial Project ideas hunt	Initial Project ideas hunt
Data extraction	Audio files and feature recognition
Project management	Documentation (50%)
Application of MLP	Application of MLP on the data
Final Report	Project Presentation

VIII. References

References

[1] https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1165context=computerscidiss
[2] https://dl.acm.org/doi/10.1007/s11042-020-09693-w
[3] Ahmad Jamil, Fiaz Mustansar, Kwon Soon-il, Sodanil Maleerat, Vo Bay, and Baik Sung
[4] Wook. 2016. Gender identification using mfcc for telephone applications-a comparative study
[5] Fusion-Based AER System Using Deep Learning Approach for Amplitude and Frequency analysis
[6] https://dl.acm.org/doi/10.1145/3488369
[7] https://www.researchgate.net/publication/
[8] https://valueml.com/multi-layer-perceptron-by-keras-with-example/
[9] https://www.hindawi.com/journals/acisc/2022/6596397/
[10] https://www.researchgate.net/publication/
[11] https://machinelearningmastery.com/build-multi-layer-perceptron-neural-network-models-keras/
[12] https://www.kaggle.com/code/sathianpong/3-ways-to-implement-mlp-with-keras/notebook
[13] https://www.turing.com/kb/multilayer-perceptron-in-tensorflow
[14] https://www.turing.com/kb/multilayer-perceptron-in-tensorflow
[15] https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/
[16] https://towardsdatascience.com/introduction-to-multilayer-neural-networks-with-tensorflows-keras-api-abf4f813959
[17] https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment
[18] https://maelfabien.github.io/deeplearning/mlp/
[19] https://machinelearninggeek.com/multi-layer-perceptron-neural-network-using-python/
[20] https://www.allaboutcircuits.com/technical-articles/how-to-create-a-multilayer-perceptron-neural-network-in-python/
[21] https://towardsdatascience.com/machine-learning-on-sound-and-audio-dhttps://www.overleaf.com/project/638e7ad1646da00940ac7852ata-3ae03bcf5095
[22] https://www.kaggle.com/code/hamditarek/audio-data-analysis-using-librosa