Code

Datasets :-

Datasets used in this project

- Crowd-sourced Emotional Mutimodal 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)

Environment :-

- Kaggle notebook
- Google colab
- Jupyter Notebook

Steps to be followed to set up an environment :

1.In this project, we select the kaggle notebook platform to run this code

2. Create a kaggle account, select a new notebook and go to settings in that turn on GPU mode choose a programming language python, set up environment as pin to original environment(2022-09-16)

3.Kaggle notebook works same as jupyter notebook .In that write code in cell & run the code

➢ Model using CNN method :-

Importing Libraries :

In [2]:

```
import pandas as pd
import numpy as np

import os
import sys
```

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

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

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

import keras
from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormaliz
ation
from keras.utils import np_utils, to_categorical
from keras.callbacks import ModelCheckpoint

import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
Using TensorFlow backend.
```

Data Preparation

- As we are working with four different datasets, so i will be creating a dataframe storing all emotions of the data in dataframe with their paths.
- We will use this dataframe to extract features for our model training.

In [3]:

```
# Paths for data.
Ravdess = "/kaggle/input/ravdess-emotional-speech-audio/audio_speech_actors_01-24/"
Crema = "/kaggle/input/cremad/AudioWAV/"
Tess = "/kaggle/input/toronto-emotional-speech-set-tess/tess toronto emotional speech
set data/TESS Toronto emotional speech set data/"
Savee = "/kaggle/input/surrey-audiovisual-expressed-emotion-savee/ALL/"
```

1. Ravdess Dataframe

Here is the filename identifiers as per the official RAVDESS website:

- 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).

So, here's an example of an audio filename. 02-01-06-01-02-01-12.mp4 This means the meta data for the audio file is:

- Video-only (02)
- Speech (01)
- Fearful (06)
- Normal intensity (01)
- Statement "dogs" (02)
- 1st Repetition (01)
- 12th Actor (12) - Female (as the actor ID number is even)

In [4]:
```
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 f
iles 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()
```
Out[4]:

	Emotions	Path
0	surprise	/kaggle/input/ravdess-emotional-speech-audio/a...
1	angry	/kaggle/input/ravdess-emotional-speech-audio/a...
2	calm	/kaggle/input/ravdess-emotional-speech-audio/a...
3	disgust	/kaggle/input/ravdess-emotional-speech-audio/a...
4	sad	/kaggle/input/ravdess-emotional-speech-audio/a...

2. Crema DataFrame

In [5]:

```
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')
```

```python
# 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()
```

Out[5]:

	Emotions	Path
0	angry	/kaggle/input/cremad/AudioWAV/1049_WSI_ANG_XX.wav
1	angry	/kaggle/input/cremad/AudioWAV/1082_IWW_ANG_XX.wav
2	fear	/kaggle/input/cremad/AudioWAV/1021_ITS_FEA_XX.wav
3	angry	/kaggle/input/cremad/AudioWAV/1086_ITS_ANG_XX.wav
4	disgust	/kaggle/input/cremad/AudioWAV/1026_ITS_DIS_XX.wav

3. TESS dataset

In [6]:

```python
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()
```

Out[6]:

	Emotions	Path
0	sad	/kaggle/input/toronto-emotional-speech-set-tes...
1	sad	/kaggle/input/toronto-emotional-speech-set-tes...
2	sad	/kaggle/input/toronto-emotional-speech-set-tes...
3	sad	/kaggle/input/toronto-emotional-speech-set-tes...
4	sad	/kaggle/input/toronto-emotional-speech-set-tes...

4. CREMA-D dataset

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

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

In [7]:

```
savee_directory_list = os.listdir(Savee)
```

```python
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()
```

Out[7]:

	Emotions	Path
0	surprise	/kaggle/input/surrey-audiovisual-expressed-emo...
1	disgust	/kaggle/input/surrey-audiovisual-expressed-emo...
2	neutral	/kaggle/input/surrey-audiovisual-expressed-emo...
3	disgust	/kaggle/input/surrey-audiovisual-expressed-emo...

	Emotions	Path
4	angry	/kaggle/input/surrey-audiovisual-expressed-emo...

In [8]:

```
# creating Dataframe using all the 4 dataframes we created so far.
data_path = pd.concat([Ravdess_df, Crema_df, Tess_df, Savee_df], axis = 0)
data_path.to_csv("data_path.csv",index=False)
data_path.head()
```

Out[8]:

	Emotions	Path
0	surprise	/kaggle/input/ravdess-emotional-speech-audio/a...
1	angry	/kaggle/input/ravdess-emotional-speech-audio/a...
2	calm	/kaggle/input/ravdess-emotional-speech-audio/a...
3	disgust	/kaggle/input/ravdess-emotional-speech-audio/a...
4	sad	/kaggle/input/ravdess-emotional-speech-audio/a...

Data Visualisation and Exploration

First let's plot the count of each emotions in our dataset.

In [9]:

```
plt.title('Count of Emotions', size=16)
sns.countplot(data_path.Emotions)
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()
```

We can also plot waveplots and spectograms for audio signals

- 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.

In [10]:

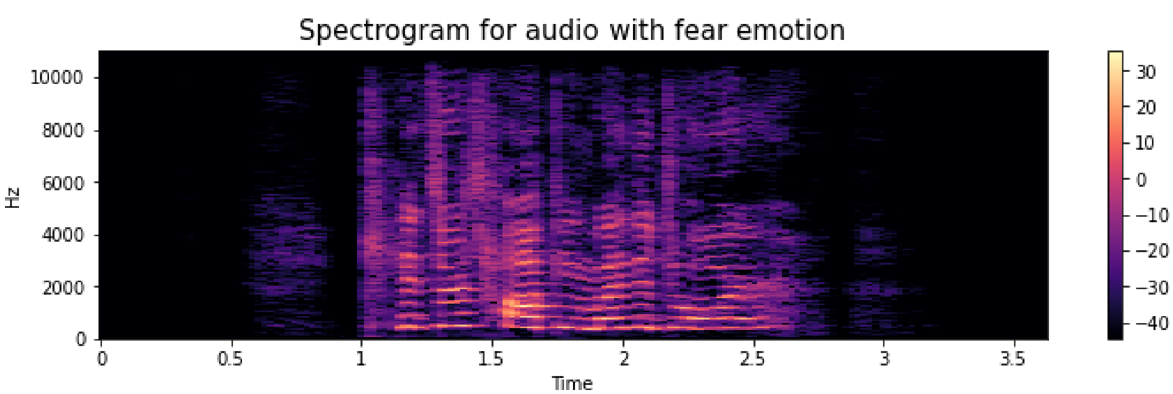
```
def create_waveplot(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for audio with {} emotion'.format(e), size=15)
    librosa.display.waveplot(data, sr=sr)
    plt.show()

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=(12, 3))
    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()
```

In [11]:

```
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)
```

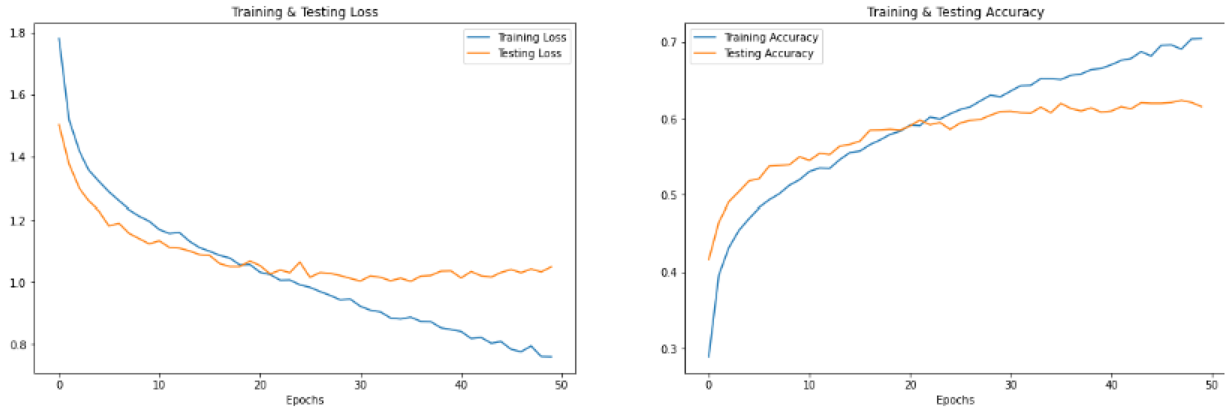
Out[11]:



In [12]:

```
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)
```

Out[12]:



In [13]:

```
emotion='sad'
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)
```

In [14]:

```
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)
```

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 augmention can be performed by shifting the image, zooming, rotating ...

In [15]:

```
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)
```

1. Simple Audio

In [16]:

```
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=data, sr=sample_rate)
Audio(path)
```

Out[16]:

2. Noise Injection

In [17]:

```
x = noise(data)
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[17]:

We can see noise injection is a very good augmentation technique because of which we can assure our training model is not overfitted

3. Stretching

In [18]:

```
x = stretch(data)
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[18]:

4. Shifting

In [19]:

```
x = shift(data)
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[19]:

5. Pitch

In [20]:

```
x = pitch(data, sample_rate)
plt.figure(figsize=(14,4))
librosa.display.waveplot(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

Out[20]:

- From the above types of augmentation techniques i am using noise, stretching(ie. changing speed) and some pitching.

Feature Extraction

- Extraction of features is a very important part in analyzing and finding relations between different things. As we already know that the data provided of 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.

1. Zero Crossing Rate : The rate of sign-changes of the signal during the duration of a particular frame.
2. Energy : The sum of squares of the signal values, normalized by the respective frame length.
3. Entropy of Energy : The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
4. Spectral Centroid : The center of gravity of the spectrum.
5. Spectral Spread : The second central moment of the spectrum.
6. Spectral Entropy : Entropy of the normalized spectral energies for a set of sub-frames.
7. Spectral Flux : The squared difference between the normalized magnitudes of the spectra of the two successive frames.
8. Spectral Rolloff : The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
9. MFCCs Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.
10. Chroma Vector : A 12-element representation of the spectral energy where the bins represent the 12 equal-tempered pitch classes of western-type music (semitone spacing).
11. 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.

In [21]:
```
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 endi
ng of each audio files as seen above.
    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 [22]:

```
X, Y = [], []
for path, emotion in zip(data_path.Path, data_path.Emotions):
    feature = get_features(path)
    for ele in feature:
        X.append(ele)
        # appending emotion 3 times as we have made 3 augmentation techniques on each
audio file.
        Y.append(emotion)
```

In [23]:

```
len(X), len(Y), data_path.Path.shape
```

Out[23]:

```
(36486, 36486, (12162,))
```

In [24]:

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

Out[24]:

	0	1	2	3	4	5	6	7	8	9	...	153	154	155	156	157	158	159	160	161	labels
0	0.185239	0.585543	0.541992	0.555859	0.615102	0.599604	0.652054	0.691854	0.766230	0.791168	...	0.002888	0.001964	0.001590	0.002071	0.002255	0.002727	0.001520	0.000461	0.000038	surprise
1	0.302097	0.748427	0.716290	0.740596	0.802801	0.760048	0.693101	0.699719	0.734826	0.753985	...	0.003670	0.002759	0.002363	0.003003	0.003083	0.003557	0.002395	0.001345	0.000886	surprise
2	0.147298	0.646143	0.595935	0.561826	0.547853	0.612391	0.561209	0.622703	0.689758	0.756473	...	0.001020	0.000665	0.000617	0.000406	0.000478	0.000603	0.000401	0.000094	0.000007	surprise
3	0.199350	0.517106	0.521565	0.508298	0.564973	0.626469	0.698655	0.668579	0.603630	0.621905	...	0.052493	0.048467	0.046119	0.036382	0.041288	0.027275	0.024452	0.006556	0.000462	angry
4	0.296762	0.653405	0.640598	0.633179	0.681640	0.741104	0.730206	0.660096	0.651581	0.663689	...	0.083794	0.079053	0.073813	0.065715	0.066659	0.054817	0.055254	0.036077	0.028982	angry

5 rows × 163 columns

- We have applied data augmentation and extracted the features for each audio files and saved them.

Data Preparation

- As of now we have extracted the data, now we need to normalize and split our data for training and testing.

In [43]:

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

In [44]:

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

In [45]:

```
# splitting data
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=0, shuffle=Tru
e)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[45]:

```
((27364, 162), (27364, 8), (9122, 162), (9122, 8))
```

In [46]:

```
# 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
```

Out[46]:

```
((27364, 162), (27364, 8), (9122, 162), (9122, 8))
```

In [47]:

```
# 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
```

Out[47]:

```
((27364, 162, 1), (27364, 8), (9122, 162, 1), (9122, 8))
```

Modelling

In [59]:

```
model=Sequential()
model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu', in
put_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 = ['ac
curacy'])

model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_28 (Conv1D)           (None, 162, 256)          1536      
_________________________________________________________________
max_pooling1d_28 (MaxPooling (None, 81, 256)           0         
_________________________________________________________________
conv1d_29 (Conv1D)           (None, 81, 256)           327936    
_________________________________________________________________
max_pooling1d_29 (MaxPooling (None, 41, 256)           0         
_________________________________________________________________
conv1d_30 (Conv1D)           (None, 41, 128)           163968    
_________________________________________________________________
max_pooling1d_30 (MaxPooling (None, 21, 128)           0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 21, 128)           0         
_________________________________________________________________
conv1d_31 (Conv1D)           (None, 21, 64)            41024     
_________________________________________________________________
max_pooling1d_31 (MaxPooling (None, 11, 64)            0         
_________________________________________________________________
flatten_7 (Flatten)          (None, 704)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 32)                22560     
_________________________________________________________________
dropout_14 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_14 (Dense)             (None, 8)                 264       
=================================================================
Total params: 557,288
Trainable params: 557,288
Non-trainable params: 0
_________________________________________________________________
```

In [60]:

```
rlrp = ReduceLROnPlateau(monitor='loss', factor=0.4, verbose=0, patience=2, min_lr=0.
0000001)
history=model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test
, y_test), callbacks=[rlrp])
Train on 27364 samples, validate on 9122 samples
Epoch 1/50
27364/27364 [==============================] - 5s 183us/step - loss: 1.6819 - acc
uracy: 0.3212 - val_loss: 1.4272 - val_accuracy: 0.4257
Epoch 2/50
```

```
27364/27364 [==============================] - 4s 163us/step - loss: 1.4340 - acc
uracy: 0.4279 - val_loss: 1.2990 - val_accuracy: 0.4752
Epoch 3/50
27364/27364 [==============================] - 4s 161us/step - loss: 1.3356 - acc
uracy: 0.4637 - val_loss: 1.2498 - val_accuracy: 0.5007
Epoch 4/50
27364/27364 [==============================] - 5s 168us/step - loss: 1.2843 - acc
uracy: 0.4928 - val_loss: 1.2138 - val_accuracy: 0.5027
Epoch 5/50
27364/27364 [==============================] - 4s 157us/step - loss: 1.2453 - acc
uracy: 0.5074 - val_loss: 1.1987 - val_accuracy: 0.5180
Epoch 6/50
27364/27364 [==============================] - 4s 163us/step - loss: 1.2134 - acc
uracy: 0.5164 - val_loss: 1.1540 - val_accuracy: 0.5406
Epoch 7/50
27364/27364 [==============================] - 4s 160us/step - loss: 1.1919 - acc
uracy: 0.5259 - val_loss: 1.1355 - val_accuracy: 0.5478
Epoch 8/50
27364/27364 [==============================] - 4s 161us/step - loss: 1.1666 - acc
uracy: 0.5315 - val_loss: 1.1249 - val_accuracy: 0.5466
Epoch 9/50
27364/27364 [==============================] - 5s 167us/step - loss: 1.1485 - acc
uracy: 0.5471 - val_loss: 1.1051 - val_accuracy: 0.5597
Epoch 10/50
27364/27364 [==============================] - 4s 159us/step - loss: 1.1272 - acc
uracy: 0.5506 - val_loss: 1.0989 - val_accuracy: 0.5628
Epoch 11/50
27364/27364 [==============================] - 5s 165us/step - loss: 1.1107 - acc
uracy: 0.5585 - val_loss: 1.0926 - val_accuracy: 0.5624
Epoch 12/50
27364/27364 [==============================] - 4s 163us/step - loss: 1.0959 - acc
uracy: 0.5676 - val_loss: 1.0999 - val_accuracy: 0.5573
Epoch 13/50
27364/27364 [==============================] - 5s 167us/step - loss: 1.0705 - acc
uracy: 0.5795 - val_loss: 1.0771 - val_accuracy: 0.5733
Epoch 14/50
27364/27364 [==============================] - 5s 170us/step - loss: 1.0616 - acc
uracy: 0.5799 - val_loss: 1.0927 - val_accuracy: 0.5704
Epoch 15/50
27364/27364 [==============================] - 4s 163us/step - loss: 1.0489 - acc
uracy: 0.5827 - val_loss: 1.0806 - val_accuracy: 0.5705
Epoch 16/50
27364/27364 [==============================] - 5s 165us/step - loss: 1.0354 - acc
uracy: 0.5871 - val_loss: 1.0807 - val_accuracy: 0.5725
Epoch 17/50
27364/27364 [==============================] - 4s 158us/step - loss: 1.0296 - acc
uracy: 0.5949 - val_loss: 1.0748 - val_accuracy: 0.5714
Epoch 18/50
27364/27364 [==============================] - 4s 157us/step - loss: 1.0165 - acc
uracy: 0.5972 - val_loss: 1.0925 - val_accuracy: 0.5640
Epoch 19/50
```

```
27364/27364 [==============================] - 5s 166us/step - loss: 0.9998 - acc
uracy: 0.6032 - val_loss: 1.0641 - val_accuracy: 0.5839
Epoch 20/50
27364/27364 [==============================] - 4s 156us/step - loss: 0.9847 - acc
uracy: 0.6125 - val_loss: 1.0481 - val_accuracy: 0.5858
Epoch 21/50
27364/27364 [==============================] - 4s 162us/step - loss: 0.9676 - acc
uracy: 0.6191 - val_loss: 1.0409 - val_accuracy: 0.5906
Epoch 22/50
27364/27364 [==============================] - 4s 156us/step - loss: 0.9564 - acc
uracy: 0.6208 - val_loss: 1.0426 - val_accuracy: 0.5932
Epoch 23/50
27364/27364 [==============================] - 4s 158us/step - loss: 0.9543 - acc
uracy: 0.6243 - val_loss: 1.0587 - val_accuracy: 0.5843
Epoch 24/50
27364/27364 [==============================] - 4s 164us/step - loss: 0.9350 - acc
uracy: 0.6343 - val_loss: 1.0419 - val_accuracy: 0.5904
Epoch 25/50
27364/27364 [==============================] - 4s 157us/step - loss: 0.9309 - acc
uracy: 0.6357 - val_loss: 1.0336 - val_accuracy: 0.5944
Epoch 26/50
27364/27364 [==============================] - 5s 170us/step - loss: 0.9138 - acc
uracy: 0.6396 - val_loss: 1.0423 - val_accuracy: 0.5932
Epoch 27/50
27364/27364 [==============================] - 5s 175us/step - loss: 0.9034 - acc
uracy: 0.6453 - val_loss: 1.0417 - val_accuracy: 0.5897
Epoch 28/50
27364/27364 [==============================] - 4s 163us/step - loss: 0.8986 - acc
uracy: 0.6473 - val_loss: 1.0485 - val_accuracy: 0.5930
Epoch 29/50
27364/27364 [==============================] - 4s 164us/step - loss: 0.8902 - acc
uracy: 0.6540 - val_loss: 1.0569 - val_accuracy: 0.5923
Epoch 30/50
27364/27364 [==============================] - 4s 157us/step - loss: 0.8712 - acc
uracy: 0.6599 - val_loss: 1.0322 - val_accuracy: 0.5996
Epoch 31/50
27364/27364 [==============================] - 5s 165us/step - loss: 0.8532 - acc
uracy: 0.6650 - val_loss: 1.0550 - val_accuracy: 0.5980
Epoch 32/50
27364/27364 [==============================] - 4s 156us/step - loss: 0.8493 - acc
uracy: 0.6661 - val_loss: 1.0356 - val_accuracy: 0.6074
Epoch 33/50
27364/27364 [==============================] - 4s 158us/step - loss: 0.8400 - acc
uracy: 0.6717 - val_loss: 1.0364 - val_accuracy: 0.6039
Epoch 34/50
27364/27364 [==============================] - 5s 165us/step - loss: 0.8331 - acc
uracy: 0.6750 - val_loss: 1.0686 - val_accuracy: 0.5956
Epoch 35/50
27364/27364 [==============================] - 4s 158us/step - loss: 0.8321 - acc
uracy: 0.6778 - val_loss: 1.0696 - val_accuracy: 0.6021
Epoch 36/50
```

```
27364/27364 [==============================] - 4s 161us/step - loss: 0.8138 - acc
uracy: 0.6814 - val_loss: 1.0885 - val_accuracy: 0.6040
Epoch 37/50
27364/27364 [==============================] - 4s 157us/step - loss: 0.8073 - acc
uracy: 0.6862 - val_loss: 1.0557 - val_accuracy: 0.6053
Epoch 38/50
27364/27364 [==============================] - 4s 156us/step - loss: 0.7902 - acc
uracy: 0.6902 - val_loss: 1.0833 - val_accuracy: 0.6030
Epoch 39/50
27364/27364 [==============================] - 4s 161us/step - loss: 0.7845 - acc
uracy: 0.6932 - val_loss: 1.0551 - val_accuracy: 0.6115
Epoch 40/50
27364/27364 [==============================] - 5s 171us/step - loss: 0.7798 - acc
uracy: 0.6967 - val_loss: 1.0646 - val_accuracy: 0.6027
Epoch 41/50
27364/27364 [==============================] - 5s 175us/step - loss: 0.7663 - acc
uracy: 0.6999 - val_loss: 1.0824 - val_accuracy: 0.6080
Epoch 42/50
27364/27364 [==============================] - 4s 164us/step - loss: 0.7606 - acc
uracy: 0.6999 - val_loss: 1.0736 - val_accuracy: 0.6095
Epoch 43/50
27364/27364 [==============================] - 4s 163us/step - loss: 0.7551 - acc
uracy: 0.7031 - val_loss: 1.0796 - val_accuracy: 0.6017
Epoch 44/50
27364/27364 [==============================] - 4s 160us/step - loss: 0.7462 - acc
uracy: 0.7140 - val_loss: 1.0945 - val_accuracy: 0.6095
Epoch 45/50
27364/27364 [==============================] - 4s 157us/step - loss: 0.7354 - acc
uracy: 0.7140 - val_loss: 1.0823 - val_accuracy: 0.6101
Epoch 46/50
27364/27364 [==============================] - 5s 164us/step - loss: 0.7295 - acc
uracy: 0.7170 - val_loss: 1.0735 - val_accuracy: 0.6084
Epoch 47/50
27364/27364 [==============================] - 4s 158us/step - loss: 0.7152 - acc
uracy: 0.7207 - val_loss: 1.0904 - val_accuracy: 0.6112
Epoch 48/50
27364/27364 [==============================] - 4s 158us/step - loss: 0.7172 - acc
uracy: 0.7233 - val_loss: 1.0881 - val_accuracy: 0.6132
Epoch 49/50
27364/27364 [==============================] - 5s 166us/step - loss: 0.6992 - acc
uracy: 0.7305 - val_loss: 1.0999 - val_accuracy: 0.6062
Epoch 50/50
27364/27364 [==============================] - 4s 157us/step - loss: 0.6977 - acc
uracy: 0.7294 - val_loss: 1.1017 - val_accuracy: 0.6074
```

In [61]:

```
print("Accuracy of our model on test data : " , model.evaluate(x_test,y_test)[1]*100
, "%")

epochs = [i for i in range(50)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
```

```
test_acc = history.history['val_accuracy']
test_loss = history.history['val_loss']

fig.set_size_inches(20,6)
ax[0].plot(epochs , train_loss , label = 'Training Loss')
ax[0].plot(epochs , test_loss , label = 'Testing Loss')
ax[0].set_title('Training & Testing Loss')
ax[0].legend()
ax[0].set_xlabel("Epochs")

ax[1].plot(epochs , train_acc , label = 'Training Accuracy')
ax[1].plot(epochs , test_acc , label = 'Testing Accuracy')
ax[1].set_title('Training & Testing Accuracy')
ax[1].legend()
ax[1].set_xlabel("Epochs")
plt.show()
```

```
9122/9122 [==============================] - 1s 92us/step
```

Accuracy of our model on test data : 60.74326038360596 %

In [62]:

```
# predicting on test data.
pred_test = model.predict(x_test)
y_pred = encoder.inverse_transform(pred_test)

y_test = encoder.inverse_transform(y_test)
```

In [63]:

```
df = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
df['Predicted Labels'] = y_pred.flatten()
df['Actual Labels'] = y_test.flatten()

df.head(10)
```

Out[63]:

	Predicted Labels	Actual Labels
0	neutral	disgust
1	sad	sad
2	sad	sad

	Predicted Labels	Actual Labels
3	fear	disgust
4	happy	happy
5	sad	fear
6	disgust	sad
7	happy	happy
8	angry	happy
9	happy	happy

In [64]:
```
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()
```

In [65]:
```
print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

       angry       0.78      0.69      0.73      1396
        calm       0.62      0.86      0.72       142
     disgust       0.54      0.48      0.51      1461
        fear       0.63      0.51      0.57      1443
```

```
       happy       0.53      0.62      0.57      1450
     neutral       0.55      0.57      0.56      1265
         sad       0.58      0.68      0.62      1470
    surprise       0.85      0.79      0.82       495

    accuracy                           0.62      9122
   macro avg       0.63      0.65      0.64      9122
weighted avg       0.61      0.61      0.61      9122
```

(2)BY USING RNN & LSTM:

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

Input data files are available in the read-only "../input/" directory

For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

for dirname, _, filenames in os.walk('/kaggle/input'):

    for filename in filenames:

        print(os.path.join(dirname, filename))

import pandas as pd

import numpy as np

Crema = "/kaggle/input/cremad/AudioWAV/"

```
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'])
```

```
path_df = pd.DataFrame(file_path, columns=['Path'])
Crema_df = pd.concat([emotion_df, path_df], axis=1)
Crema_df.head()

plt.title('Count of Emotions', size=16)
sns.countplot(Crema_df.Emotions)
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()

def create_waveshow(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for {} emotion'.format(e), size=15)
    librosa.display.waveshow(data, sr=sr)
    plt.show()

def create_spectrogram(data, sr, e):
    X = librosa.stft(data)
    Xdb = librosa.amplitude_to_db(abs(X))
    plt.figure(figsize=(12, 3))
    plt.title('Spectrogram for {} emotion'.format(e), size=15)
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
    plt.colorbar()

emotion='angry'
path = np.array(Crema_df.Path[Crema_df.Emotions==emotion])[0]
```

```
data, sampling_rate = librosa.load(path)
create_waveshow(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
num_mfcc=13
n_fft=2048
hop_length=512
SAMPLE_RATE = 22050
data = {
    "labels": [],
    "mfcc": []
  }
for i in range(7442):
  data['labels'].append(Crema_df.iloc[i,0])
  signal, sample_rate = librosa.load(Crema_df.iloc[i,1], sr=SAMPLE_RATE)
  mfcc = librosa.feature.mfcc(signal, sample_rate, n_mfcc=13, n_fft=2048, hop_length=512)
  mfcc = mfcc.T
  data["mfcc"].append(np.asarray(mfcc))
  if i%500==0:
    print(i)
```

output :-

```
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
```

6000
6500
7000

```
X = np.asarray(data['mfcc'])

y = np.asarray(data["labels"])


X = tf.keras.preprocessing.sequence.pad_sequences(X)

X.shape


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)

X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=0.2)


print(X_train.shape,y_train.shape,X_validation.shape,y_validation.shape,X_test.shape,y_test.sh
ape)


def build_model(input_shape):
    model = tf.keras.Sequential()


    model.add(LSTM(128, input_shape=input_shape, return_sequences=True))
    model.add(LSTM(64))


    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.3))


    model.add(Dense(6, activation='softmax'))


    return model
```

create network

```
input_shape = (None,13)
model = build_model(input_shape)

# compile model
optimiser = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimiser,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

output:

```
2022-09-18 07:08:18.246676: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool wit
h default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
lstm (LSTM)                 (None, None, 128)         72704
_________________________________________________________________
lstm_1 (LSTM)               (None, 64)                49408
_________________________________________________________________
dense (Dense)               (None, 64)                4160
_________________________________________________________________
dropout (Dropout)           (None, 64)                0
_________________________________________________________________
dense_1 (Dense)             (None, 6)                 390
=================================================================
Total params: 126,662
Trainable params: 126,662
Non-trainable params: 0
_________________________________________________________________

history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=32, epochs=30)
2022-09-18 07:08:26.262088: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
```

Epoch 1/30
168/168 [==============================] - 74s 412ms/step - loss: 1.5750 - accuracy: 0.3405 - val_loss: 1
.4950 - val_accuracy: 0.3761
Epoch 2/30
168/168 [==============================] - 69s 413ms/step - loss: 1.4889 - accuracy: 0.3862 - val_loss: 1
.4592 - val_accuracy: 0.4149
Epoch 3/30
168/168 [==============================] - 67s 399ms/step - loss: 1.4412 - accuracy: 0.4094 - val_loss: 1
.4062 - val_accuracy: 0.4090
Epoch 4/30
168/168 [==============================] - 67s 399ms/step - loss: 1.4250 - accuracy: 0.4181 - val_loss: 1
.3974 - val_accuracy: 0.4216
Epoch 5/30
168/168 [==============================] - 66s 392ms/step - loss: 1.4061 - accuracy: 0.4181 - val_loss: 1
.3710 - val_accuracy: 0.4440
Epoch 6/30
168/168 [==============================] - 66s 394ms/step - loss: 1.3759 - accuracy: 0.4353 - val_loss: 1
.3620 - val_accuracy: 0.4358
Epoch 7/30
168/168 [==============================] - 68s 408ms/step - loss: 1.3686 - accuracy: 0.4476 - val_loss: 1
.3747 - val_accuracy: 0.4231
Epoch 8/30
168/168 [==============================] - 68s 406ms/step - loss: 1.3453 - accuracy: 0.4491 - val_loss: 1
.3447 - val_accuracy: 0.4545
Epoch 9/30
168/168 [==============================] - 67s 401ms/step - loss: 1.3228 - accuracy: 0.4609 - val_loss: 1
.3493 - val_accuracy: 0.4791
Epoch 10/30
168/168 [==============================] - 67s 400ms/step - loss: 1.3083 - accuracy: 0.4753 - val_loss: 1
.3279 - val_accuracy: 0.4627
Epoch 11/30
168/168 [==============================] - 68s 406ms/step - loss: 1.2853 - accuracy: 0.4874 - val_loss: 1
.3176 - val_accuracy: 0.4701
Epoch 12/30
168/168 [==============================] - 69s 412ms/step - loss: 1.2741 - accuracy: 0.4971 - val_loss: 1
.2767 - val_accuracy: 0.5149
Epoch 13/30
168/168 [==============================] - 67s 399ms/step - loss: 1.2519 - accuracy: 0.5173 - val_loss: 1
.2585 - val_accuracy: 0.5060
Epoch 14/30
168/168 [==============================] - 68s 403ms/step - loss: 1.2178 - accuracy: 0.5214 - val_loss: 1
.2332 - val_accuracy: 0.5261
Epoch 15/30
168/168 [==============================] - 66s 393ms/step - loss: 1.1983 - accuracy: 0.5350 - val_loss: 1
.2392 - val_accuracy: 0.5313
Epoch 16/30
168/168 [==============================] - 66s 396ms/step - loss: 1.1638 - accuracy: 0.5529 - val_loss: 1
.2156 - val_accuracy: 0.5306
Epoch 17/30
168/168 [==============================] - 68s 402ms/step - loss: 1.1598 - accuracy: 0.5555 - val_loss: 1
.2283 - val_accuracy: 0.5276
Epoch 18/30
168/168 [==============================] - 68s 404ms/step - loss: 1.1541 - accuracy: 0.5509 - val_loss: 1
.2615 - val_accuracy: 0.5104
Epoch 19/30

```
168/168 [==============================] - 66s 396ms/step - loss: 1.1334 - accuracy: 0.5615 - val_loss: 1
.2159 - val_accuracy: 0.5381
Epoch 20/30
168/168 [==============================] - 67s 397ms/step - loss: 1.0859 - accuracy: 0.5906 - val_loss: 1
.2271 - val_accuracy: 0.5246
Epoch 21/30
168/168 [==============================] - 70s 415ms/step - loss: 1.0753 - accuracy: 0.5942 - val_loss: 1
.1762 - val_accuracy: 0.5604
Epoch 22/30
168/168 [==============================] - 68s 402ms/step - loss: 1.0646 - accuracy: 0.5888 - val_loss: 1
.1804 - val_accuracy: 0.5470
Epoch 23/30
168/168 [==============================] - 69s 409ms/step - loss: 1.0408 - accuracy: 0.6069 - val_loss: 1
.2377 - val_accuracy: 0.5537
Epoch 24/30
168/168 [==============================] - 67s 399ms/step - loss: 1.0345 - accuracy: 0.6095 - val_loss: 1
.1853 - val_accuracy: 0.5649
Epoch 25/30
168/168 [==============================] - 67s 400ms/step - loss: 1.0012 - accuracy: 0.6231 - val_loss: 1
.1708 - val_accuracy: 0.5761
Epoch 26/30
168/168 [==============================] - 68s 405ms/step - loss: 0.9774 - accuracy: 0.6295 - val_loss: 1
.1922 - val_accuracy: 0.5507
Epoch 27/30
168/168 [==============================] - 67s 398ms/step - loss: 0.9836 - accuracy: 0.6280 - val_loss: 1
.2524 - val_accuracy: 0.5500
Epoch 28/30
168/168 [==============================] - 68s 403ms/step - loss: 0.9758 - accuracy: 0.6323 - val_loss: 1
.1464 - val_accuracy: 0.5627
Epoch 29/30
168/168 [==============================] - 68s 406ms/step - loss: 0.9349 - accuracy: 0.6487 - val_loss: 1
.1604 - val_accuracy: 0.5769
Epoch 30/30
168/168 [==============================] - 69s 408ms/step - loss: 0.9133 - accuracy: 0.6606 - val_loss: 1
.1630 - val_accuracy: 0.570
```

```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)

print("Test Accuracy: ",test_acc)
```

<u>output:-</u>

```
Test Accuracy:  0.5758389234542847
```

(3)

```
# This Python 3 environment comes with many helpful analytics libraries installed
```

```
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdir("../input"))
for df in ("../input"):
   df=pd.read_csv("../input/preprocessing.csv").fillna(0)

# Any results you write to the current directory are saved as output.
df.head()
df.info()
```

<u>output:-</u>

```
class 'pandas.core.frame.DataFrame'>
RangeIndex: 329 entries, 0 to 328
Columns: 106 entries, ID to MFCC_104
dtypes: float64(104), object(2)
memory usage: 272.5+ KB
```

```
df.corr()
print("Total number of labels: {}".format(df.shape[0]))
```

output:-

```
Total number of labels: 329
```

```
X.dtypes.sample(104)
```

Output:-

```
MFCC_59    float64
MFCC_93    float64
MFCC_50    float64
MFCC_39    float64
MFCC_18    float64
MFCC_96    float64
MFCC_28    float64
MFCC_76    float64
MFCC_13    float64
MFCC_77    float64
MFCC_15    float64
MFCC_56    float64
MFCC_44    float64
MFCC_69    float64
MFCC_1     float64
MFCC_46    float64
MFCC_31    float64
MFCC_37    float64
MFCC_75    float64
MFCC_34    float64
MFCC_52    float64
MFCC_11    float64
MFCC_90    float64
MFCC_36    float64
MFCC_24    float64
MFCC_99    float64
MFCC_57    float64
MFCC_72    float64
MFCC_65    float64
MFCC_10    float64
            ...
MFCC_4     float64
MFCC_80    float64
MFCC_48    float64
MFCC_23    float64
MFCC_33    float64
MFCC_22    float64
MFCC_89    float64
MFCC_60    float64
MFCC_79    float64
MFCC_7     float64
MFCC_51    float64
MFCC_97    float64
MFCC_42    float64
```

```
MFCC_9     float64
MFCC_85    float64
MFCC_20    float64
MFCC_38    float64
MFCC_49    float64
MFCC_91    float64
MFCC_30    float64
MFCC_27    float64
MFCC_62    float64
MFCC_58    float64
MFCC_6     float64
MFCC_12    float64
MFCC_2     float64
MFCC_74    float64
MFCC_67    float64
MFCC_14    float64
MFCC_29    float64
Length: 104, dtype: object
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 0)

from sklearn.linear_model import LogisticRegression
m1 = LogisticRegression()
m1.fit(X_train, y_train)
pred1 = m1.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, pred1))
```

<u>output:-</u>

```
precision      recall     f1-score    support

     ANGER    0.64      0.33      0.44       21
     FEAR      0.33      0.33      0.33        3
     HAPPY      0.14      0.10      0.12       10
     NEUTRAL  0.65      0.58      0.61       19
      SAD          0.31      0.71      0.43        7
    SURPRISE   0.17      0.33      0.22        6

avg / total       0.47      0.41      0.41       66
```

```
pred2 = m2.predict(X_test)
print(classification_report(y_test, pred2)) #much better, but recall is still low
```