Speech_Emotion_Recognition(2)

May 5, 2024

[ ]:
```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'ravdess-emotional-speech-audio:https%3A%2F%2Fstorage.
↪googleapis.com%2Fkaggle-data-sets%2F107620%2F256618%2Fbundle%2Farchive.
↪zip%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-1
↪iam.gserviceaccount.
↪com%252F20240504%252Fauto%252Fstorage%252Fgoog4_request%26X-Goog-Date%3D20240504T144813Z%26
↪https%3A%2F%2Fstorage.googleapis.
↪com%2Fkaggle-data-sets%2F316368%2F639622%2Fbundle%2Farchive.
↪zip%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-1
↪iam.gserviceaccount.
↪com%252F20240504%252Fauto%252Fstorage%252Fgoog4_request%26X-Goog-Date%3D20240504T144813Z%26
↪https%3A%2F%2Fstorage.googleapis.
↪com%2Fkaggle-data-sets%2F325566%2F653195%2Fbundle%2Farchive.
↪zip%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-1
↪iam.gserviceaccount.
↪com%252F20240504%252Fauto%252Fstorage%252Fgoog4_request%26X-Goog-Date%3D20240504T144813Z%26
↪https%3A%2F%2Fstorage.googleapis.
↪com%2Fkaggle-data-sets%2F338555%2F671851%2Fbundle%2Farchive.
↪zip%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-1
↪iam.gserviceaccount.
↪com%252F20240504%252Fauto%252Fstorage%252Fgoog4_request%26X-Goog-Date%3D20240504T144813Z%26

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
```

```
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
```

```
        continue

print('Data source import complete.')
```

```
Downloading ravdess-emotional-speech-audio, 450102890 bytes compressed
[==================================================] 450102890 bytes downloaded
Downloaded and uncompressed: ravdess-emotional-speech-audio
Downloading toronto-emotional-speech-set-tess, 448572034 bytes compressed
[==================================================] 448572034 bytes downloaded
Downloaded and uncompressed: toronto-emotional-speech-set-tess
Downloading cremad, 473324524 bytes compressed
[==================================================] 473324524 bytes downloaded
Downloaded and uncompressed: cremad
Downloading surrey-audiovisual-expressed-emotion-savee, 112690765 bytes
compressed
[==================================================] 112690765 bytes downloaded
Downloaded and uncompressed: surrey-audiovisual-expressed-emotion-savee
Data source import complete.
```

#

Speech Emotion Recognition

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)

1 Importing Libraries

[ ]:
```
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
 ↪extract 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,␣
 ↪BatchNormalization
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint

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

1.1 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.

[ ]:
```
# 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)

[ ]:
```
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()
```

[ ]:
```
   Emotions                                               Path
0   neutral  /kaggle/input/ravdess-emotional-speech-audio/a…
1      fear  /kaggle/input/ravdess-emotional-speech-audio/a…
2  surprise  /kaggle/input/ravdess-emotional-speech-audio/a…
3     happy  /kaggle/input/ravdess-emotional-speech-audio/a…
4      fear  /kaggle/input/ravdess-emotional-speech-audio/a…
```

##

2. Crema DataFrame

[ ]:
```
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()
```

[ ]:

	Emotions	Path
0	angry	/kaggle/input/cremad/AudioWAV/1056_IWW_ANG_XX.wav
1	disgust	/kaggle/input/cremad/AudioWAV/1055_IWL_DIS_XX.wav
2	fear	/kaggle/input/cremad/AudioWAV/1025_IEO_FEA_MD.wav
3	sad	/kaggle/input/cremad/AudioWAV/1055_IEO_SAD_MD.wav
4	sad	/kaggle/input/cremad/AudioWAV/1052_IOM_SAD_XX.wav

##

3. TESS dataset

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

file_emotion = []
file_path = []
```

```python
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()
```

[ ]:

	Emotions	Path
0	surprise	/kaggle/input/toronto-emotional-speech-set-tes…
1	surprise	/kaggle/input/toronto-emotional-speech-set-tes…
2	surprise	/kaggle/input/toronto-emotional-speech-set-tes…
3	surprise	/kaggle/input/toronto-emotional-speech-set-tes…
4	surprise	/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'

[ ]:

```python
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()
```

[ ]:

	Emotions	Path
0	angry	/kaggle/input/surrey-audiovisual-expressed-emo…
1	surprise	/kaggle/input/surrey-audiovisual-expressed-emo…
2	neutral	/kaggle/input/surrey-audiovisual-expressed-emo…
3	surprise	/kaggle/input/surrey-audiovisual-expressed-emo…
4	fear	/kaggle/input/surrey-audiovisual-expressed-emo…

[ ]:

```
# 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()
```

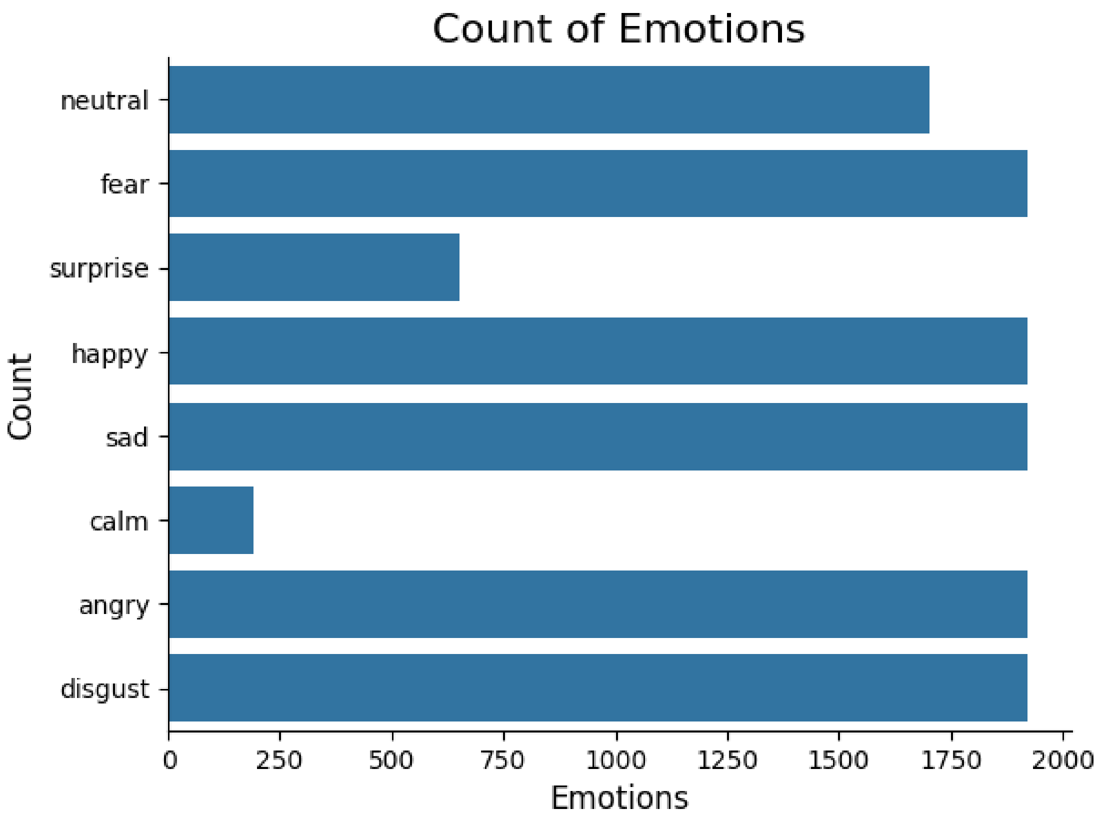
[ ]:

	Emotions	Path
0	neutral	/kaggle/input/ravdess-emotional-speech-audio/a…
1	fear	/kaggle/input/ravdess-emotional-speech-audio/a…
2	surprise	/kaggle/input/ravdess-emotional-speech-audio/a…
3	happy	/kaggle/input/ravdess-emotional-speech-audio/a…
4	fear	/kaggle/input/ravdess-emotional-speech-audio/a…

1.2 Data Visualisation and Exploration

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

[ ]:
```
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()
```



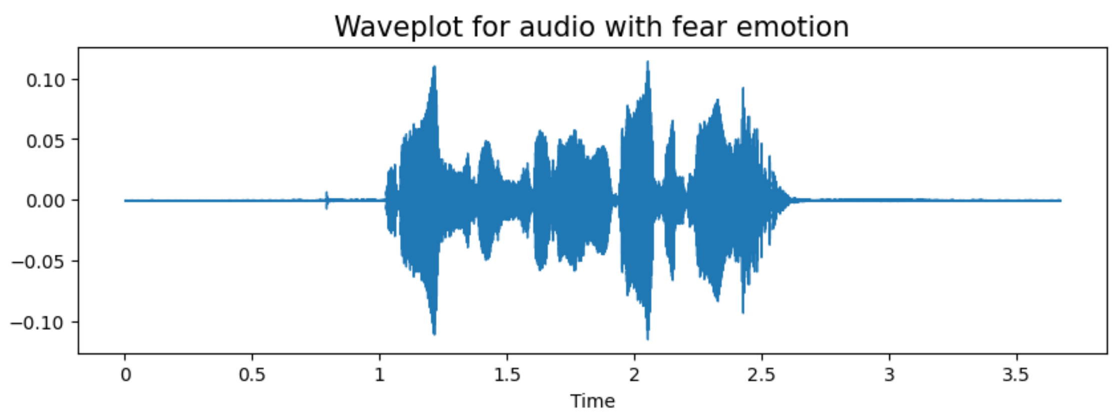
We can also plot waveplots and spectograms for audio signals

- 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.

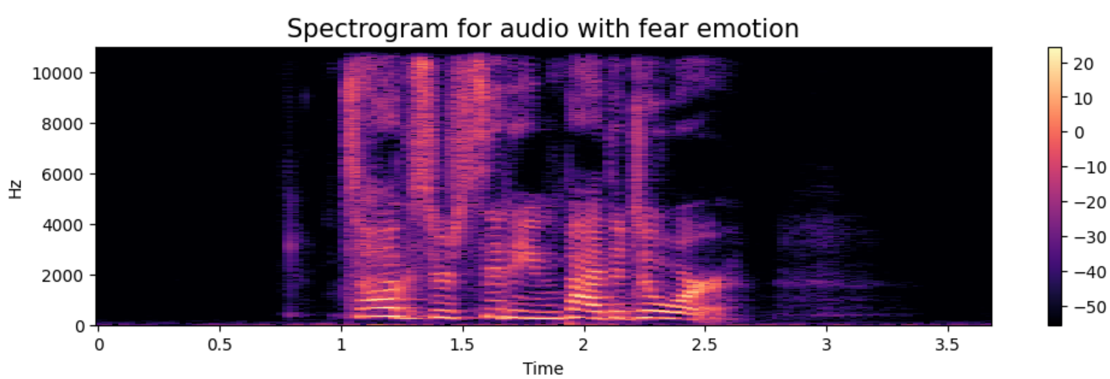
[ ]:
```
def create_waveplot(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for audio with {} emotion'.format(e), size=15)
    librosa.display.waveshow(data, sr=sampling_rate)
    plt.show()

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()
```

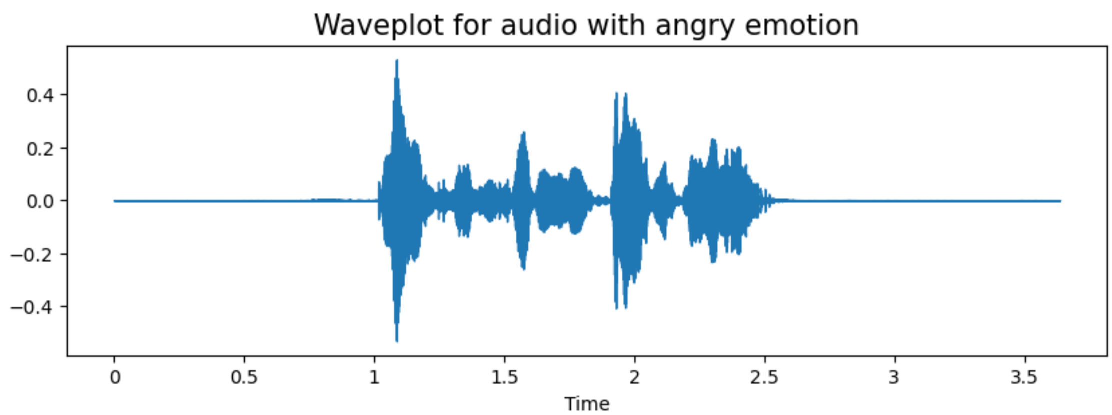
[ ]:
```
emotion='fear'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



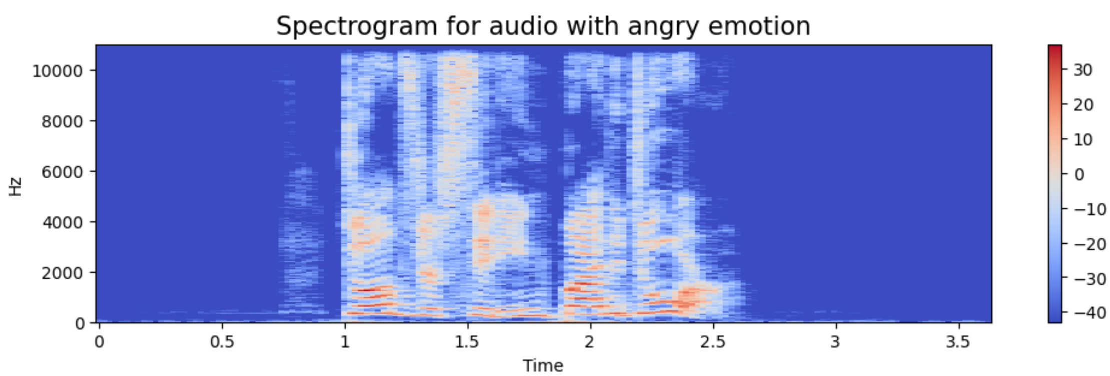
[ ]: <IPython.lib.display.Audio object>



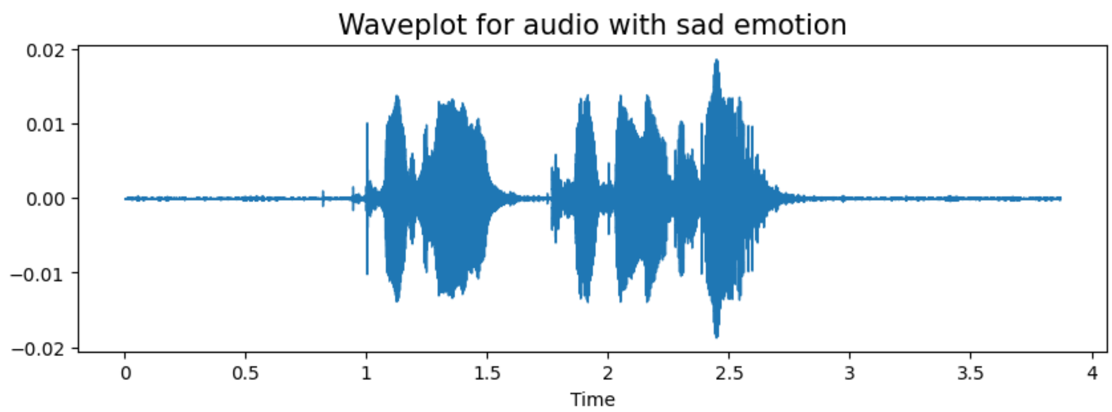
```
[ ]: emotion='angry'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



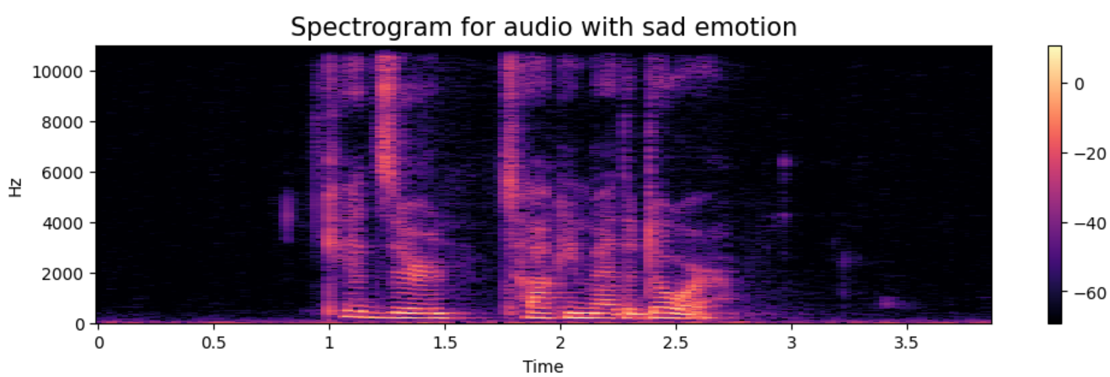

```
[ ]: <IPython.lib.display.Audio object>
```



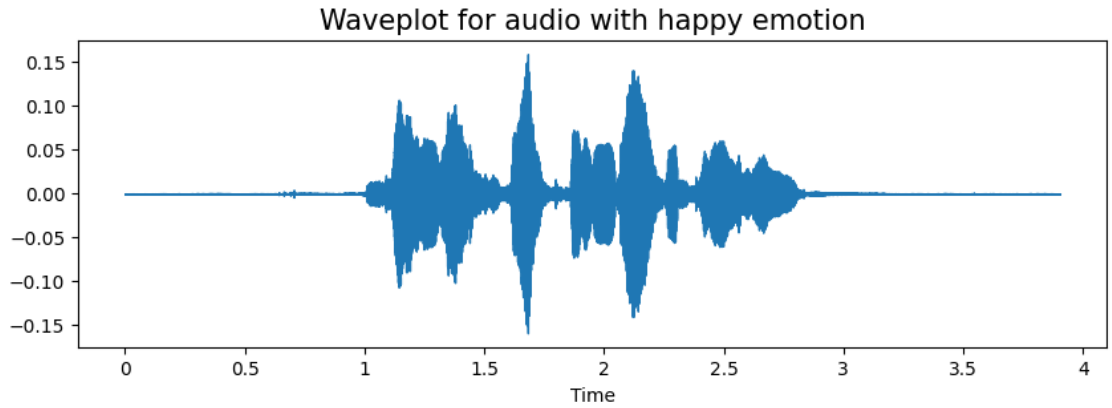

```
[ ]: emotion='sad'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



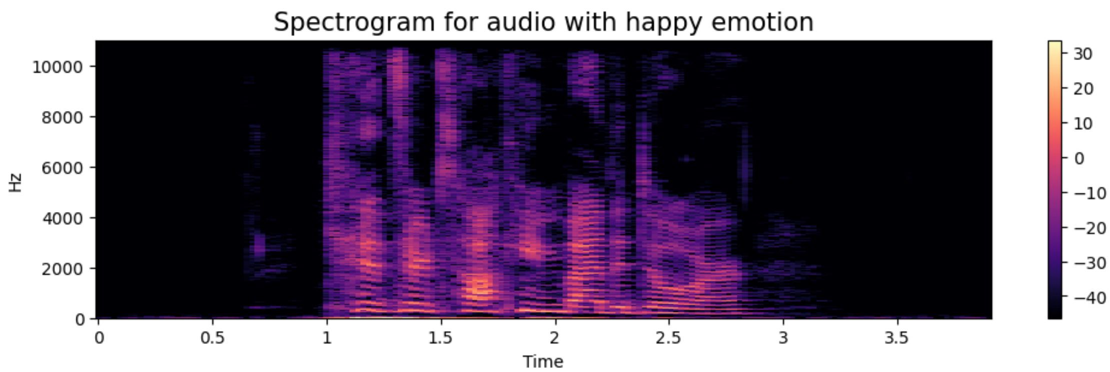
[ ]: <IPython.lib.display.Audio object>



[ ]:
```
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)
```



[ ]: <IPython.lib.display.Audio object>



1.3 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 …

First, let's check which augmentation techniques works better for our dataset.

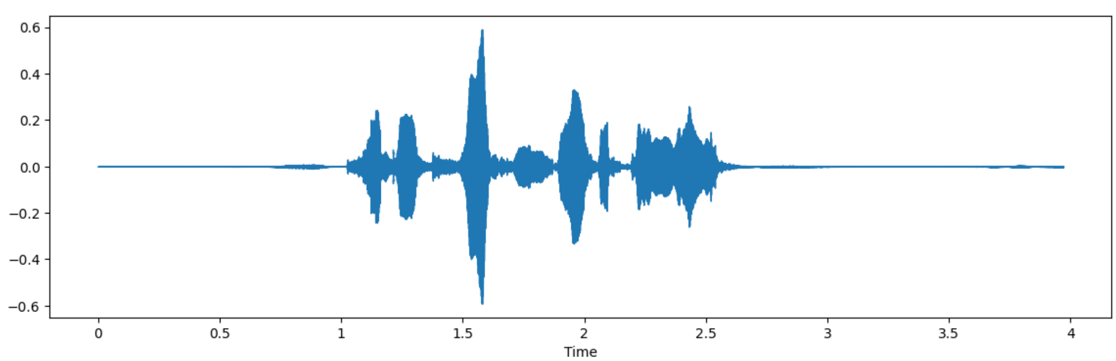
[ ]:
```
def noise(data):
    noise_amp = 0.035*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
```

```python
    return data

def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate=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):
    n_steps = int(np.random.uniform(low=-5, high=5))
    return librosa.effects.pitch_shift(data, sr=sampling_rate, n_steps=n_steps)

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

1. Simple Audio

[ ]:
```python
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=data, sr=sample_rate)
Audio(path)
```

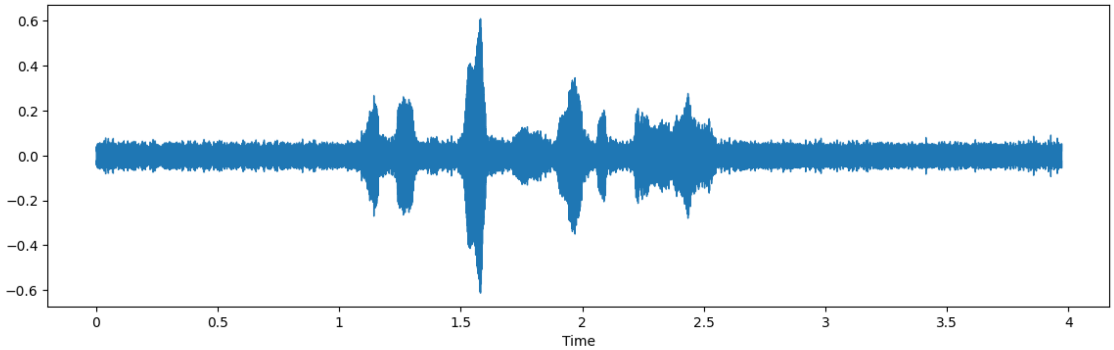
[ ]: <IPython.lib.display.Audio object>



2. Noise Injection

[ ]:
```python
x = noise(data)
plt.figure(figsize=(14,4))
librosa.display.waveshow(y=x, sr=sample_rate)
Audio(x, rate=sample_rate)
```

[ ]: <IPython.lib.display.Audio object>

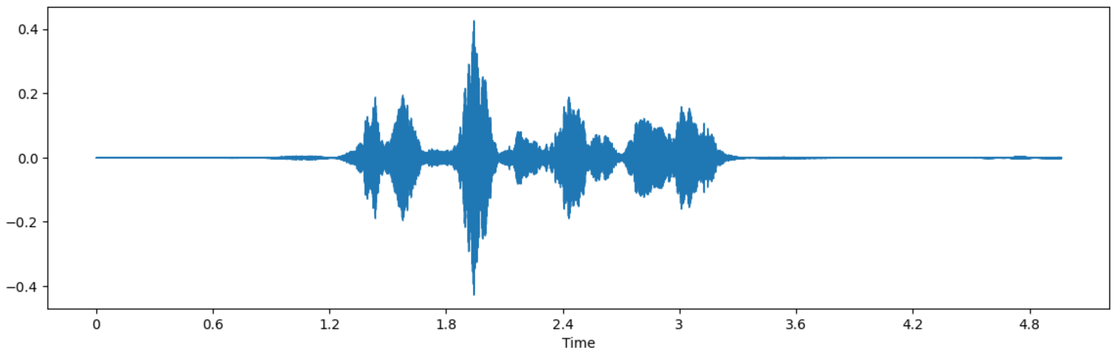


We can see noise injection is a very good augmentation technique because of which we can assure our training model is not overfitted

3. Stretching

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

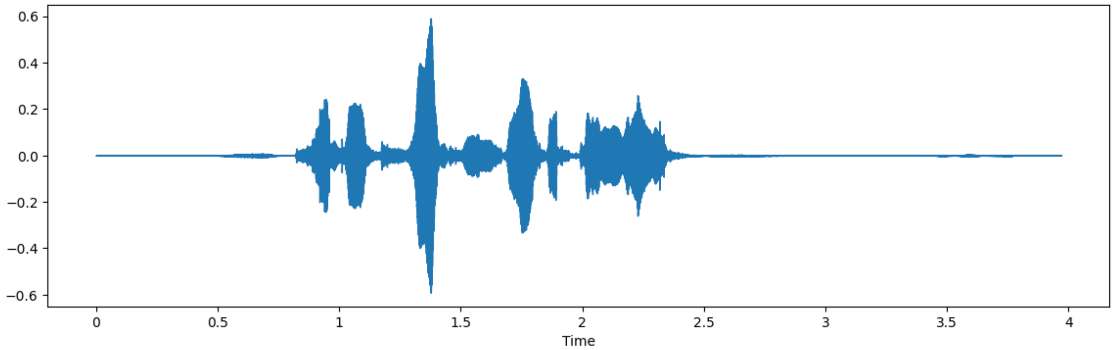
[ ]: <IPython.lib.display.Audio object>



4. Shifting

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

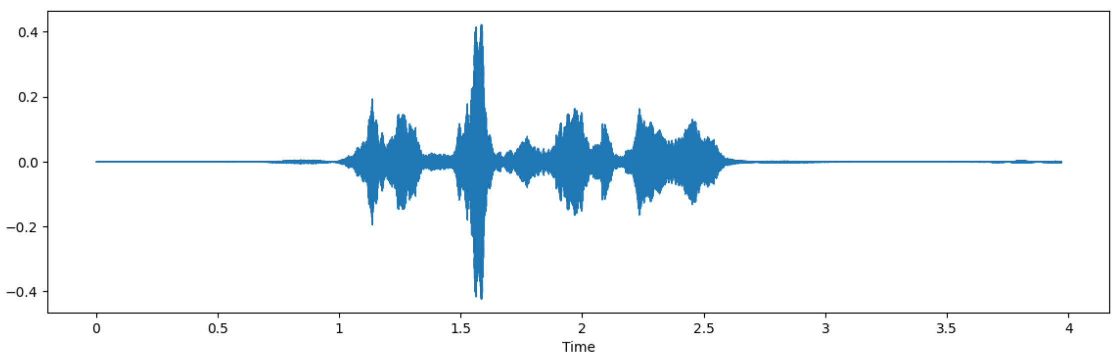
[ ]: <IPython.lib.display.Audio object>



5. Pitch

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

```
[ ]: <IPython.lib.display.Audio object>
```



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

1.4 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.

```
[ ]: 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 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
```

[ ]:
```python
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)
```

```
/usr/local/lib/python3.10/dist-packages/librosa/core/pitch.py:101: UserWarning:
Trying to estimate tuning from empty frequency set.
  return pitch_tuning(
```

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

[ ]:
```
(36486, 36486, (12162,))
```

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

[ ]:
```
          0         1         2         3         4         5         6  \
0  0.148532  0.510419  0.501593  0.547143  0.620452  0.621838  0.567530
1  0.262071  0.592838  0.603497  0.596738  0.628165  0.680699  0.668271
2  0.128447  0.496892  0.489544  0.500267  0.546271  0.590283  0.607705
3  0.166974  0.478113  0.457538  0.455389  0.472822  0.488651  0.548206
4  0.293737  0.609718  0.624298  0.640169  0.645290  0.668220  0.671542

          7         8         9  …       153       154       155       156  \
0  0.586726  0.648204  0.711573  …  0.000166  0.000052  0.000037  0.000056
1  0.592070  0.627646  0.688654  …  0.000190  0.000076  0.000058  0.000081
2  0.598313  0.566849  0.629814  …  0.000031  0.000036  0.000041  0.000009
3  0.690389  0.735753  0.686422  …  0.000755  0.001030  0.000923  0.001115
4  0.718024  0.709053  0.700008  …  0.003802  0.004010  0.003948  0.004455

        157       158       159       160           161   labels
0  0.000069  0.000081  0.000062  0.000013  8.081029e-07  neutral
1  0.000093  0.000102  0.000082  0.000035  2.167632e-05  neutral
2  0.000007  0.000008  0.000011  0.000009  9.180562e-07  neutral
3  0.001090  0.001110  0.001362  0.000512  3.914442e-05     fear
4  0.004187  0.004137  0.004341  0.003812  3.111563e-03     fear

[5 rows x 163 columns]
```

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

1.5 Data Preparation

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

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

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

[ ]:
```
# 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
```

[ ]:
```
((27364, 162), (27364, 8), (9122, 162), (9122, 8))
```

[ ]:
```
# 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
```

[ ]:
```
((27364, 162), (27364, 8), (9122, 162), (9122, 8))
```

[ ]:
```
# 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
```

[ ]:
```
((27364, 162, 1), (27364, 8), (9122, 162, 1), (9122, 8))
```

1.6 Modelling

[ ]:
```
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()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 162, 256)          1536

 max_pooling1d (MaxPooling1  (None, 81, 256)           0
 D)

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

 max_pooling1d_1 (MaxPoolin  (None, 41, 256)           0
 g1D)

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

 max_pooling1d_2 (MaxPoolin  (None, 21, 128)           0
 g1D)

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

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

 max_pooling1d_3 (MaxPoolin  (None, 11, 64)            0
 g1D)

 flatten (Flatten)           (None, 704)               0

 dense (Dense)               (None, 32)                22560
```

```
 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 8)                 264

=================================================================
Total params: 557288 (2.13 MB)
Trainable params: 557288 (2.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
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])
```

```
Epoch 1/50
428/428 [==============================] - 177s 408ms/step - loss: 1.6819 -
accuracy: 0.3224 - val_loss: 1.4493 - val_accuracy: 0.4266 - lr: 0.0010
Epoch 2/50
428/428 [==============================] - 171s 399ms/step - loss: 1.4664 -
accuracy: 0.4157 - val_loss: 1.3752 - val_accuracy: 0.4536 - lr: 0.0010
Epoch 3/50
428/428 [==============================] - 177s 414ms/step - loss: 1.3823 -
accuracy: 0.4462 - val_loss: 1.3314 - val_accuracy: 0.4620 - lr: 0.0010
Epoch 4/50
428/428 [==============================] - 166s 388ms/step - loss: 1.3361 -
accuracy: 0.4686 - val_loss: 1.2616 - val_accuracy: 0.4901 - lr: 0.0010
Epoch 5/50
428/428 [==============================] - 178s 416ms/step - loss: 1.2988 -
accuracy: 0.4825 - val_loss: 1.2605 - val_accuracy: 0.4942 - lr: 0.0010
Epoch 6/50
428/428 [==============================] - 168s 393ms/step - loss: 1.2751 -
accuracy: 0.4946 - val_loss: 1.2333 - val_accuracy: 0.5091 - lr: 0.0010
Epoch 7/50
428/428 [==============================] - 179s 417ms/step - loss: 1.2427 -
accuracy: 0.5051 - val_loss: 1.2161 - val_accuracy: 0.5175 - lr: 0.0010
Epoch 8/50
428/428 [==============================] - 190s 443ms/step - loss: 1.2270 -
accuracy: 0.5125 - val_loss: 1.2229 - val_accuracy: 0.5068 - lr: 0.0010
Epoch 9/50
428/428 [==============================] - 169s 395ms/step - loss: 1.2035 -
accuracy: 0.5225 - val_loss: 1.1657 - val_accuracy: 0.5286 - lr: 0.0010
Epoch 10/50
428/428 [==============================] - 176s 411ms/step - loss: 1.1833 -
accuracy: 0.5304 - val_loss: 1.1631 - val_accuracy: 0.5374 - lr: 0.0010
Epoch 11/50
428/428 [==============================] - 175s 408ms/step - loss: 1.1773 -
accuracy: 0.5286 - val_loss: 1.1685 - val_accuracy: 0.5383 - lr: 0.0010
```

```
Epoch 12/50
428/428 [==============================] - 168s 393ms/step - loss: 1.1596 -
accuracy: 0.5370 - val_loss: 1.1386 - val_accuracy: 0.5491 - lr: 0.0010
Epoch 13/50
428/428 [==============================] - 176s 412ms/step - loss: 1.1482 -
accuracy: 0.5395 - val_loss: 1.1436 - val_accuracy: 0.5456 - lr: 0.0010
Epoch 14/50
428/428 [==============================] - 174s 408ms/step - loss: 1.1384 -
accuracy: 0.5484 - val_loss: 1.1327 - val_accuracy: 0.5501 - lr: 0.0010
Epoch 15/50
428/428 [==============================] - 174s 406ms/step - loss: 1.1196 -
accuracy: 0.5559 - val_loss: 1.1357 - val_accuracy: 0.5463 - lr: 0.0010
Epoch 16/50
428/428 [==============================] - 175s 408ms/step - loss: 1.1036 -
accuracy: 0.5623 - val_loss: 1.1562 - val_accuracy: 0.5401 - lr: 0.0010
Epoch 17/50
428/428 [==============================] - 174s 406ms/step - loss: 1.1075 -
accuracy: 0.5556 - val_loss: 1.1181 - val_accuracy: 0.5535 - lr: 0.0010
Epoch 18/50
428/428 [==============================] - 174s 406ms/step - loss: 1.0905 -
accuracy: 0.5696 - val_loss: 1.1165 - val_accuracy: 0.5562 - lr: 0.0010
Epoch 19/50
428/428 [==============================] - 173s 405ms/step - loss: 1.0843 -
accuracy: 0.5684 - val_loss: 1.1351 - val_accuracy: 0.5494 - lr: 0.0010
Epoch 20/50
428/428 [==============================] - 166s 388ms/step - loss: 1.0708 -
accuracy: 0.5740 - val_loss: 1.1019 - val_accuracy: 0.5648 - lr: 0.0010
Epoch 21/50
428/428 [==============================] - 175s 408ms/step - loss: 1.0646 -
accuracy: 0.5786 - val_loss: 1.1303 - val_accuracy: 0.5571 - lr: 0.0010
Epoch 22/50
428/428 [==============================] - 172s 403ms/step - loss: 1.0457 -
accuracy: 0.5864 - val_loss: 1.0953 - val_accuracy: 0.5699 - lr: 0.0010
Epoch 23/50
428/428 [==============================] - 172s 401ms/step - loss: 1.0416 -
accuracy: 0.5880 - val_loss: 1.1203 - val_accuracy: 0.5580 - lr: 0.0010
Epoch 24/50
428/428 [==============================] - 164s 382ms/step - loss: 1.0241 -
accuracy: 0.5952 - val_loss: 1.1090 - val_accuracy: 0.5617 - lr: 0.0010
Epoch 25/50
428/428 [==============================] - 169s 394ms/step - loss: 1.0306 -
accuracy: 0.5932 - val_loss: 1.0998 - val_accuracy: 0.5671 - lr: 0.0010
Epoch 26/50
428/428 [==============================] - 175s 408ms/step - loss: 1.0117 -
accuracy: 0.6040 - val_loss: 1.1068 - val_accuracy: 0.5657 - lr: 0.0010
Epoch 27/50
428/428 [==============================] - 172s 401ms/step - loss: 1.0000 -
accuracy: 0.6023 - val_loss: 1.1137 - val_accuracy: 0.5602 - lr: 0.0010
```

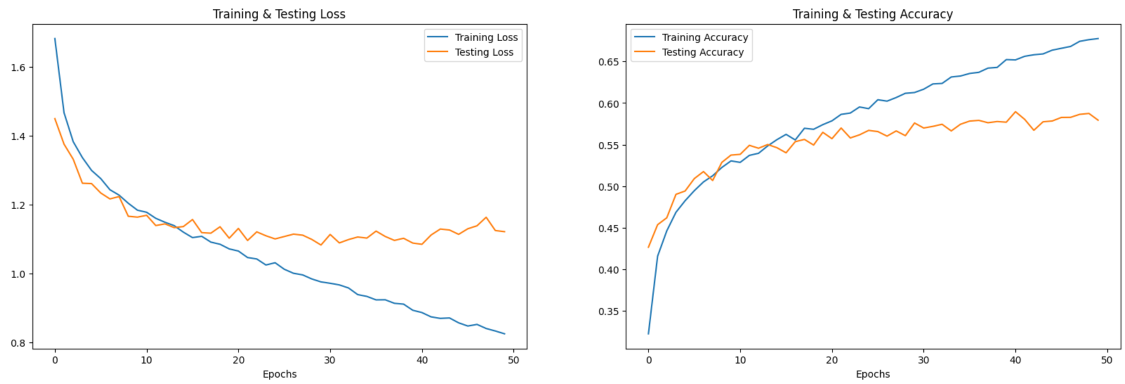
```
Epoch 28/50
428/428 [==============================] - 174s 407ms/step - loss: 0.9953 -
accuracy: 0.6066 - val_loss: 1.1108 - val_accuracy: 0.5664 - lr: 0.0010
Epoch 29/50
428/428 [==============================] - 175s 409ms/step - loss: 0.9835 -
accuracy: 0.6117 - val_loss: 1.0983 - val_accuracy: 0.5607 - lr: 0.0010
Epoch 30/50
428/428 [==============================] - 166s 388ms/step - loss: 0.9750 -
accuracy: 0.6126 - val_loss: 1.0821 - val_accuracy: 0.5760 - lr: 0.0010
Epoch 31/50
428/428 [==============================] - 174s 406ms/step - loss: 0.9710 -
accuracy: 0.6166 - val_loss: 1.1128 - val_accuracy: 0.5699 - lr: 0.0010
Epoch 32/50
428/428 [==============================] - 163s 382ms/step - loss: 0.9662 -
accuracy: 0.6230 - val_loss: 1.0882 - val_accuracy: 0.5719 - lr: 0.0010
Epoch 33/50
428/428 [==============================] - 173s 405ms/step - loss: 0.9573 -
accuracy: 0.6236 - val_loss: 1.0979 - val_accuracy: 0.5744 - lr: 0.0010
Epoch 34/50
428/428 [==============================] - 172s 403ms/step - loss: 0.9381 -
accuracy: 0.6313 - val_loss: 1.1054 - val_accuracy: 0.5664 - lr: 0.0010
Epoch 35/50
428/428 [==============================] - 165s 386ms/step - loss: 0.9328 -
accuracy: 0.6325 - val_loss: 1.1022 - val_accuracy: 0.5744 - lr: 0.0010
Epoch 36/50
428/428 [==============================] - 174s 407ms/step - loss: 0.9227 -
accuracy: 0.6355 - val_loss: 1.1225 - val_accuracy: 0.5783 - lr: 0.0010
Epoch 37/50
428/428 [==============================] - 173s 405ms/step - loss: 0.9230 -
accuracy: 0.6369 - val_loss: 1.1070 - val_accuracy: 0.5791 - lr: 0.0010
Epoch 38/50
428/428 [==============================] - 172s 403ms/step - loss: 0.9128 -
accuracy: 0.6420 - val_loss: 1.0955 - val_accuracy: 0.5763 - lr: 0.0010
Epoch 39/50
428/428 [==============================] - 165s 386ms/step - loss: 0.9104 -
accuracy: 0.6428 - val_loss: 1.1017 - val_accuracy: 0.5777 - lr: 0.0010
Epoch 40/50
428/428 [==============================] - 177s 414ms/step - loss: 0.8926 -
accuracy: 0.6522 - val_loss: 1.0878 - val_accuracy: 0.5770 - lr: 0.0010
Epoch 41/50
428/428 [==============================] - 175s 409ms/step - loss: 0.8859 -
accuracy: 0.6519 - val_loss: 1.0842 - val_accuracy: 0.5896 - lr: 0.0010
Epoch 42/50
428/428 [==============================] - 175s 410ms/step - loss: 0.8733 -
accuracy: 0.6562 - val_loss: 1.1112 - val_accuracy: 0.5804 - lr: 0.0010
Epoch 43/50
428/428 [==============================] - 176s 412ms/step - loss: 0.8690 -
accuracy: 0.6581 - val_loss: 1.1285 - val_accuracy: 0.5672 - lr: 0.0010
```

```
Epoch 44/50
428/428 [==============================] - 165s 386ms/step - loss: 0.8703 -
accuracy: 0.6590 - val_loss: 1.1258 - val_accuracy: 0.5774 - lr: 0.0010
Epoch 45/50
428/428 [==============================] - 173s 404ms/step - loss: 0.8560 -
accuracy: 0.6636 - val_loss: 1.1131 - val_accuracy: 0.5784 - lr: 0.0010
Epoch 46/50
428/428 [==============================] - 174s 407ms/step - loss: 0.8467 -
accuracy: 0.6658 - val_loss: 1.1294 - val_accuracy: 0.5827 - lr: 0.0010
Epoch 47/50
428/428 [==============================] - 174s 406ms/step - loss: 0.8515 -
accuracy: 0.6680 - val_loss: 1.1380 - val_accuracy: 0.5828 - lr: 0.0010
Epoch 48/50
428/428 [==============================] - 176s 411ms/step - loss: 0.8396 -
accuracy: 0.6742 - val_loss: 1.1628 - val_accuracy: 0.5864 - lr: 0.0010
Epoch 49/50
428/428 [==============================] - 174s 406ms/step - loss: 0.8323 -
accuracy: 0.6762 - val_loss: 1.1241 - val_accuracy: 0.5875 - lr: 0.0010
Epoch 50/50
428/428 [==============================] - 167s 391ms/step - loss: 0.8242 -
accuracy: 0.6775 - val_loss: 1.1209 - val_accuracy: 0.5794 - lr: 0.0010
```

[ ]:
```
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()
```

```
286/286 [==============================] - 13s 44ms/step - loss: 1.1209 -
accuracy: 0.5794
```

Accuracy of our model on test data : 57.9368531703949 %



[ ]:
```
# 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 [==============================] - 17s 58ms/step
```

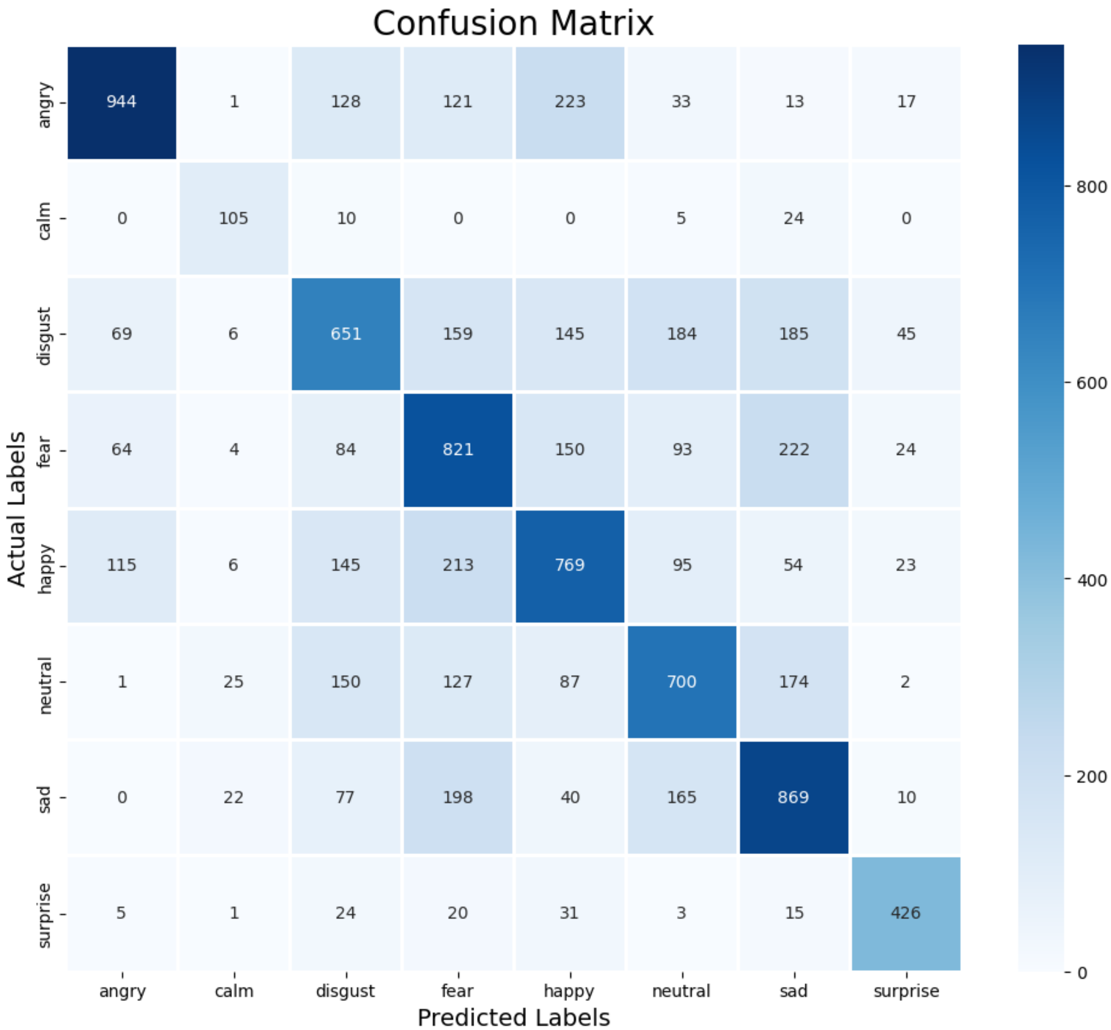
[ ]:
```
df = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
df['Predicted Labels'] = y_pred.flatten()
df['Actual Labels'] = y_test.flatten()

df.head(10)
```

[ ]:

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

[ ]:
```
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()
```



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

```
              precision    recall  f1-score   support

       angry       0.79      0.64      0.71      1480
        calm       0.62      0.73      0.67       144
     disgust       0.51      0.45      0.48      1444
        fear       0.49      0.56      0.53      1462
       happy       0.53      0.54      0.54      1420
     neutral       0.55      0.55      0.55      1266
```

```
         sad       0.56      0.63      0.59      1381
    surprise       0.78      0.81      0.79       525

    accuracy                           0.58      9122
   macro avg       0.60      0.61      0.61      9122
weighted avg       0.59      0.58      0.58      9122
```

- We can see our model is more accurate in predicting surprise, angry emotions and it makes sense also because audio files of these emotions differ to other audio files in a lot of ways like pitch, speed etc..
- We overall achieved 61% accuracy on our test data and its decent but we can improve it more by applying more augmentation techniques and using other feature extraction methods.