```
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 np_utils, to_categorical
from keras.callbacks import ModelCheckpoint

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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
Ravdess = "/content/drive/MyDrive/t9h6p943xy-5/BanglaSER"
```

```
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:
        #print(file)
        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'])
banser = pd.concat([emotion_df, path_df], axis=1)

# changing integers to actual emotions.
banser.Emotions.replace({1:'happy', 2:'sad', 3:'angry', 4:'surprise', 5:'neutral'}, inplace=True)
banser.head()
```

	Emotions	Path
0	sad	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
1	sad	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
2	happy	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
3	surprise	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
4	surprise	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...

```
banser.describe
```

```
<bound method NDFrame.describe of       Emotions                                              Path
0          sad  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
1          sad  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
2        happy  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
3     surprise  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
4     surprise  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
...        ...                                               ...
1462   neutral  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
```

```
1463     angry  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
1464  surprise  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
1465     angry  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
1466   neutral  /content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...

[1467 rows x 2 columns]>
```

```
#!unzip /content/drive/MyDrive/SUBESCO.zip -d /content/drive/MyDrive/
```

```
dataset2 = '/content/drive/MyDrive/SUBESCO/'
```

```
dataset2_list = os.listdir(dataset2)
```

```
uniq=[]
```

```
for file in dataset2_list:
    # storing file paths
    file_path.append(dataset2 + file)
    # storing file emotions
    part=file.split('_')
    uniq.append(part[5])
```

```
data2_emo=pd.DataFrame(uniq,columns= ['names'])
```

```
data2_emo.names.unique()
```

```
array(['NEUTRAL', 'HAPPY', 'SURPRISE', 'SAD', 'DISGUST', 'ANGRY', 'FEAR'],
      dtype=object)
```

```
file_emotion2 = []
file_path2 = []
```

```
for file in dataset2_list:
    # storing file paths
    file_path2.append(dataset2 + file)
    # storing file emotions
    part=file.split('_')
    if part[5] == 'SAD':
        file_emotion2.append('sad')
    elif part[5] == 'ANGRY':
        file_emotion2.append('angry')
    elif part[5] == 'HAPPY':
        file_emotion2.append('happy')
    elif part[5] == 'NEUTRAL':
        file_emotion2.append('neutral')
    elif part[5] == 'SURPRISE':
        file_emotion2.append('surprise')

# dataframe for emotion of files
emotion_df2 = pd.DataFrame(file_emotion2, columns=['Emotions'])

# dataframe for path of files.
path_df2 = pd.DataFrame(file_path2, columns=['Path'])
subesco = pd.concat([emotion_df2, path_df2], axis=1)
subesco.head()
```

	Emotions	Path
0	neutral	/content/drive/MyDrive/SUBESCO/F_07_TITHI_S_2_...
1	neutral	/content/drive/MyDrive/SUBESCO/F_07_TITHI_S_2_...
2	happy	/content/drive/MyDrive/SUBESCO/F_07_TITHI_S_2_...
3	neutral	/content/drive/MyDrive/SUBESCO/F_07_TITHI_S_2_...
4	neutral	/content/drive/MyDrive/SUBESCO/F_07_TITHI_S_2_...

```
subesco.Emotions.unique()
```

```
array(['neutral', 'happy', 'surprise', 'sad', 'angry', nan], dtype=object)
```

```
banser.Emotions.unique()
```

```
array(['sad', 'happy', 'surprise', 'angry', 'neutral'], dtype=object)
```

```
subesco.describe()
```

	Emotions	Path
count	5000	7000
unique	5	7000
top	neutral	/content/drive/MyDrive/SUBESCO/F_07_TITHI_S_2_...

```
banser.describe()
```

	Emotions	Path
count	1467	1467
unique	5	1467
top	sad	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
freq	306	1

```
dataset_path = pd.concat([banser, subesco], axis = 0)
dataset_path.to_csv("Project_Dataset.csv",index=False)
dataset_path.head()
```

	Emotions	Path
0	sad	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
1	sad	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
2	happy	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
3	surprise	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
4	surprise	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...

```
Features = pd.read_csv('/content/drive/MyDrive/New_Features.csv')
Features.dropna(axis=0, inplace=True)
```

```
<ipython-input-4-3cb90927c5a8>:1: DtypeWarning: Columns (58) have mixed types. Specify dtype option on import or set low_
  Features = pd.read_csv('/content/drive/MyDrive/New_Features.csv')
```

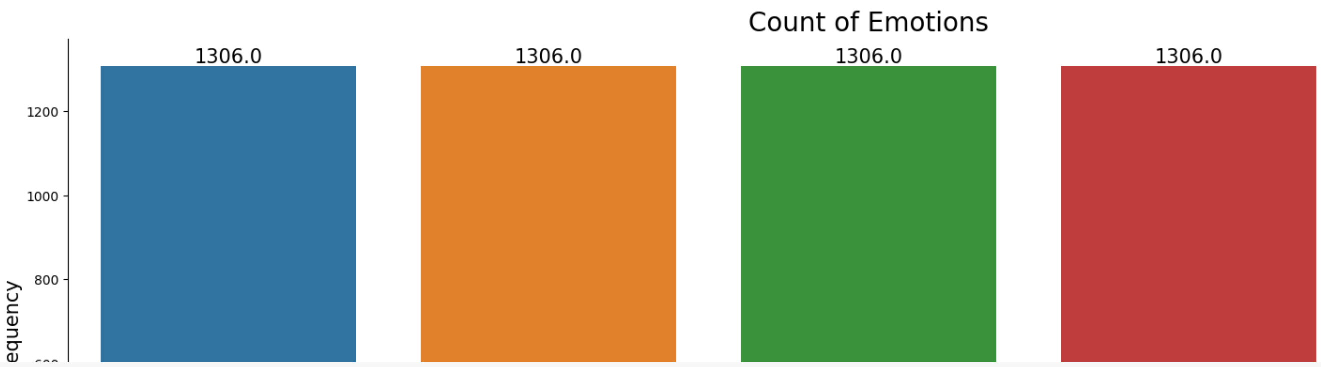
```
dataset_path.describe()
```

	Emotions	Path
count	6467	8467
unique	5	8467
top	sad	/content/drive/MyDrive/t9h6p943xy-5/BanglaSER/...
freq	1306	1

```
dataset_path.Emotions.unique()
```

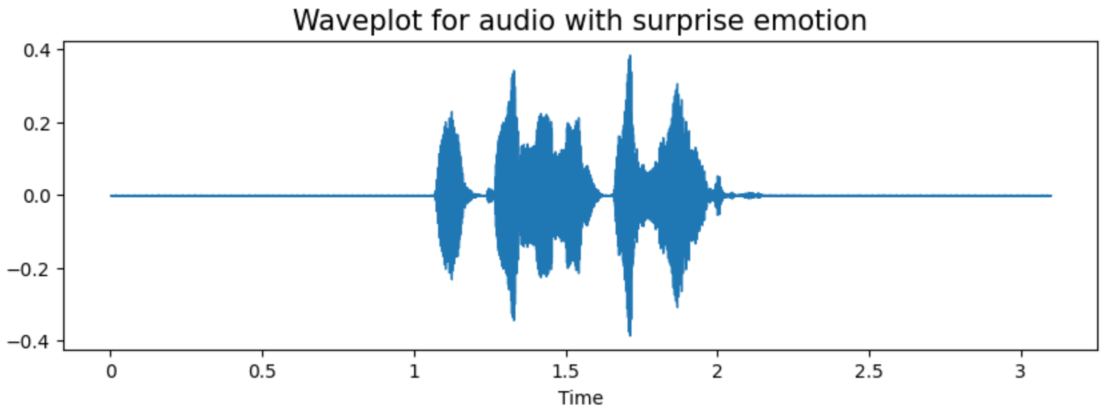
```
array(['sad', 'happy', 'surprise', 'angry', 'neutral', nan], dtype=object)
```

```
plt.figure(figsize=(22, 8))
ax = sns.countplot(x=dataset_path.Emotions)

# Add total count labels to each bar
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2, height, height, ha='center', va='bottom',size = 15)

plt.ylabel('Frequency', size=15)
plt.xlabel('Emotions', size=15)
plt.title('Count of Emotions', size=20)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()
```

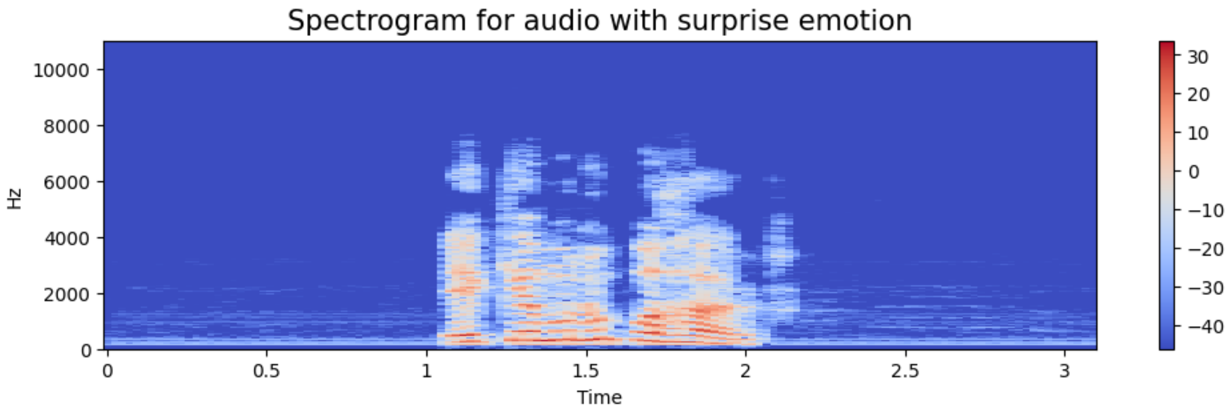


```
def create_waveplot(data, sr, e):
    plt.figure(figsize=(10, 3))
    plt.title('Waveplot for audio with {} emotion'.format(e), size=15)
    librosa.display.waveshow(data, sr=sr)
    plt.show()

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')
    plt.colorbar()
```

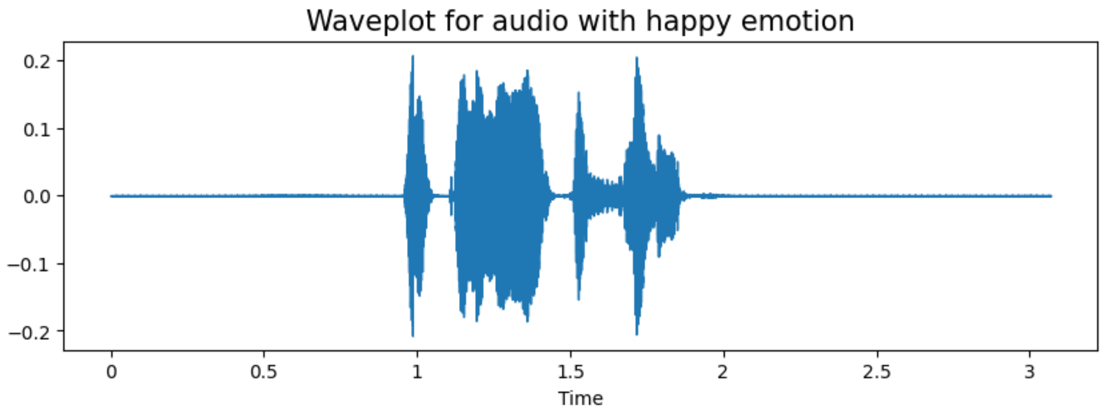
```
emotion='surprise'
path = np.array(dataset_path.Path[dataset_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



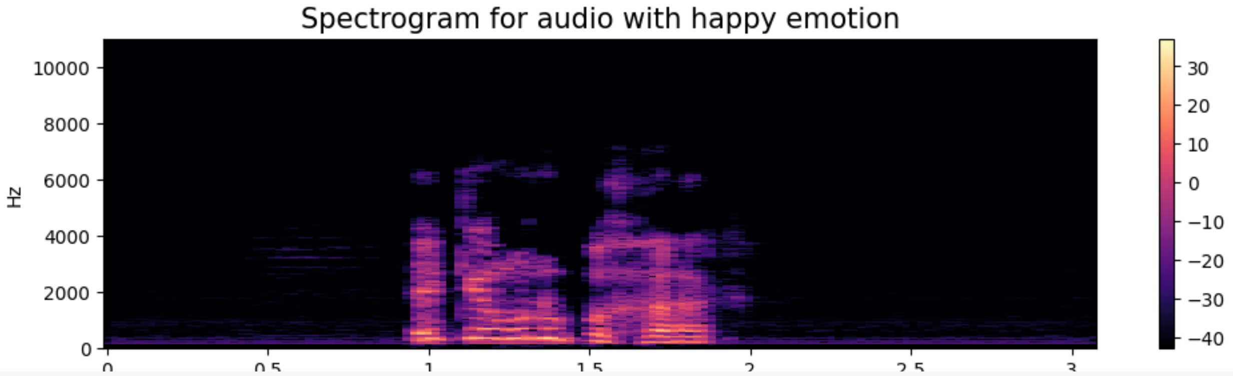
0:00 / 0:03



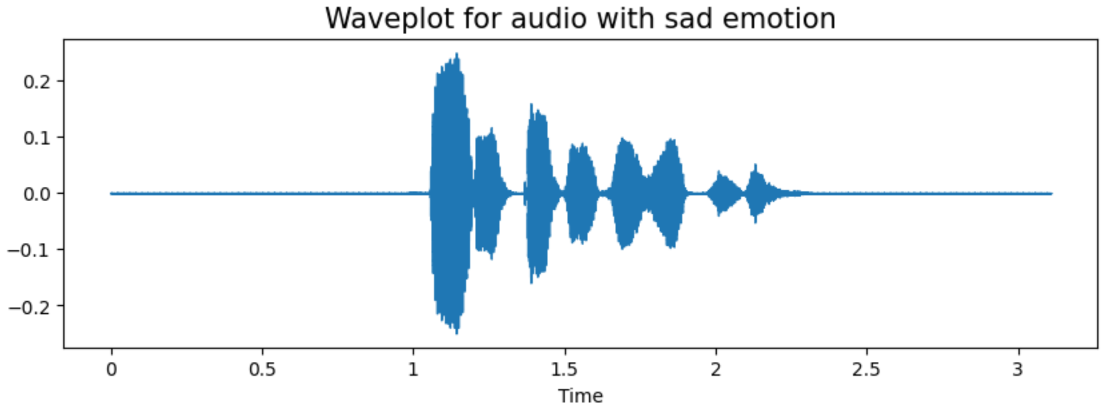
```
emotion='happy'
path = np.array(dataset_path.Path[dataset_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



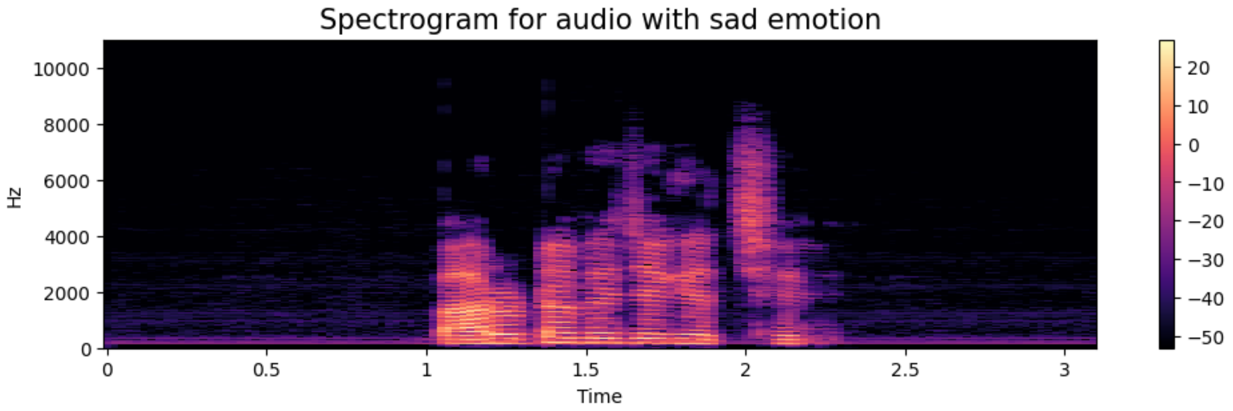
0:00 / 0:03



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

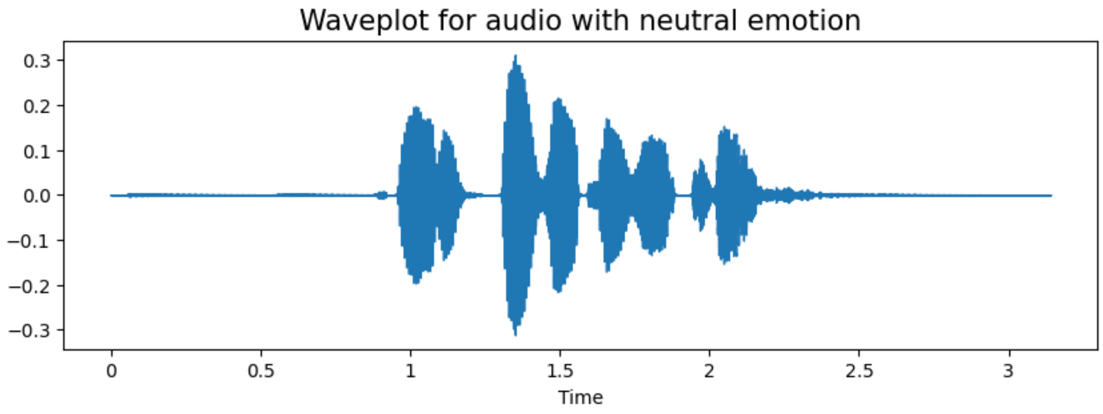


0:00 / 0:03

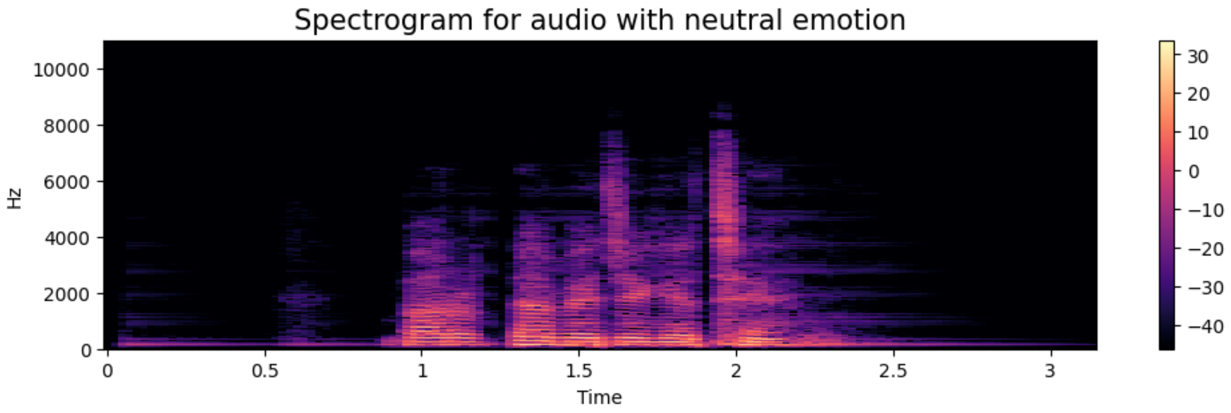


```
emotion='neutral'
path = np.array(dataset_path.Path[dataset_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
```

```
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

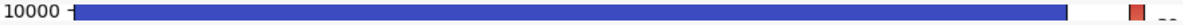


0:00 / 0:03



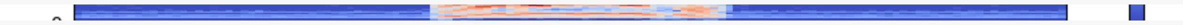
```
emotion='angry'
path = np.array(dataset_path.Path[dataset_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```

```
def noise(data):
    data = np.array(data, dtype=float)  # Convert data to a numeric array
    noise_amp = 0.04 * np.random.uniform() * np.amax(data)
    data = data + noise_amp * np.random.normal(size=data.shape[0])
    return data

def stretch(data, rate=0.70):
    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):
    return librosa.effects.pitch_shift(data, sr=sampling_rate, n_steps=pitch_factor)

def higher_speed(data, speed_factor = 1.25):
    return librosa.effects.time_stretch(data,rate=speed_factor)

def lower_speed(data, speed_factor = 0.75):
    return librosa.effects.time_stretch(data,rate=speed_factor)
```

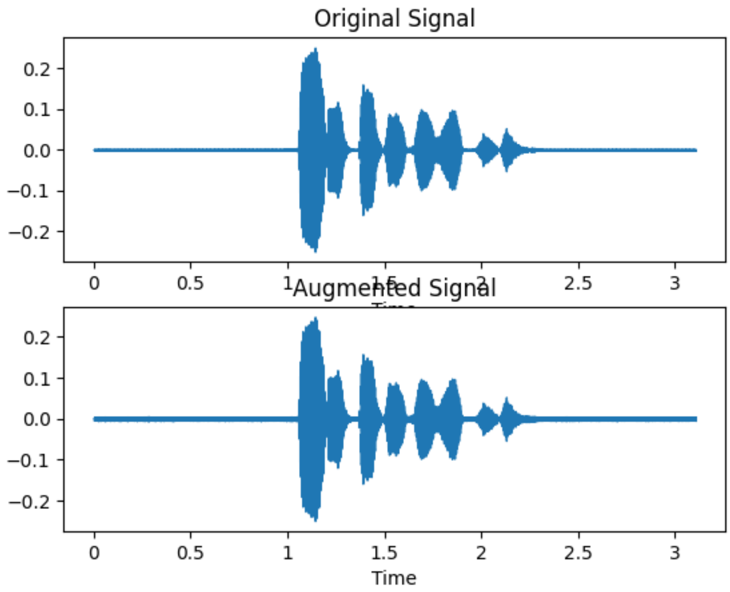


```
dataset_path
```

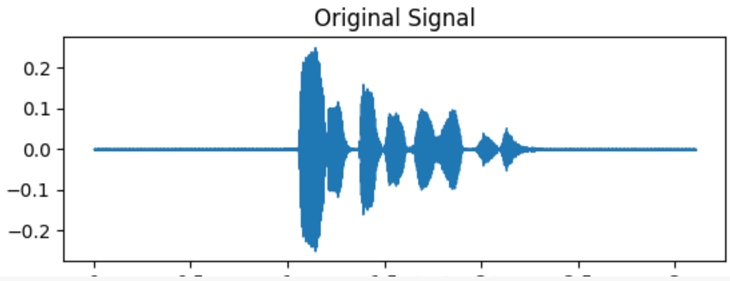
```
def _plot_signal_and_augmented_signal(signal, augmented_signal, sr):
    fig, ax = plt.subplots(nrows=2)
    librosa.display.waveshow(signal, sr=sr, ax=ax[0])
    ax[0].set(title='Original Signal')
    librosa.display.waveshow(augmented_signal, sr=sr, ax=ax[1])
    ax[1].set(title='Augmented Signal')
    plt.show()
```



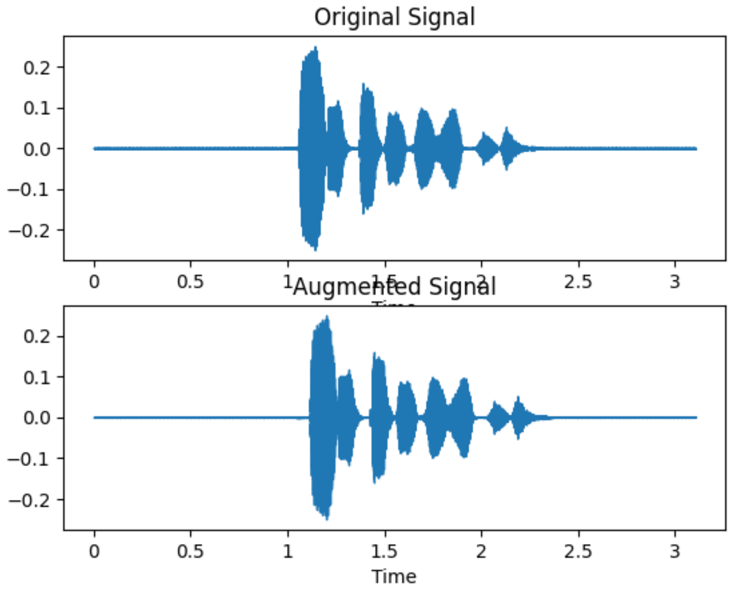
```
path = np.array(dataset_path.Path)[1]
data, sr = librosa.load(path)
augmented_signal = noise(data)
_plot_signal_and_augmented_signal(data, augmented_signal, sr)
```



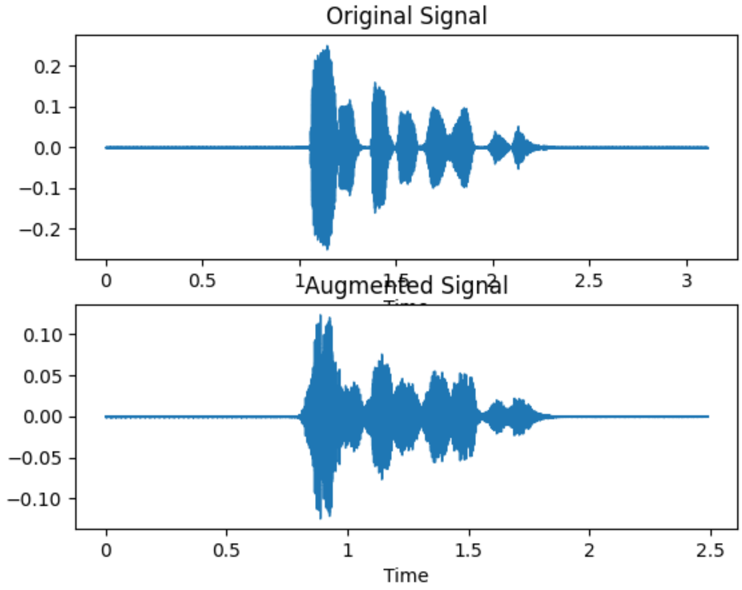
```
path = np.array(dataset_path.Path)[1]
data, sr = librosa.load(path)
augmented_signal = stretch(data)
_plot_signal_and_augmented_signal(data, augmented_signal, sr)
```



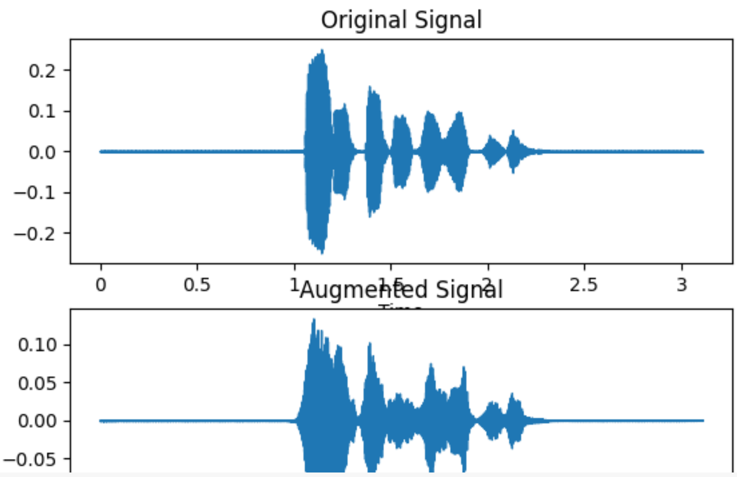
```
path = np.array(dataset_path.Path)[1]
data, sr = librosa.load(path)
augmented_signal = shift(data)
_plot_signal_and_augmented_signal(data, augmented_signal, sr)
```



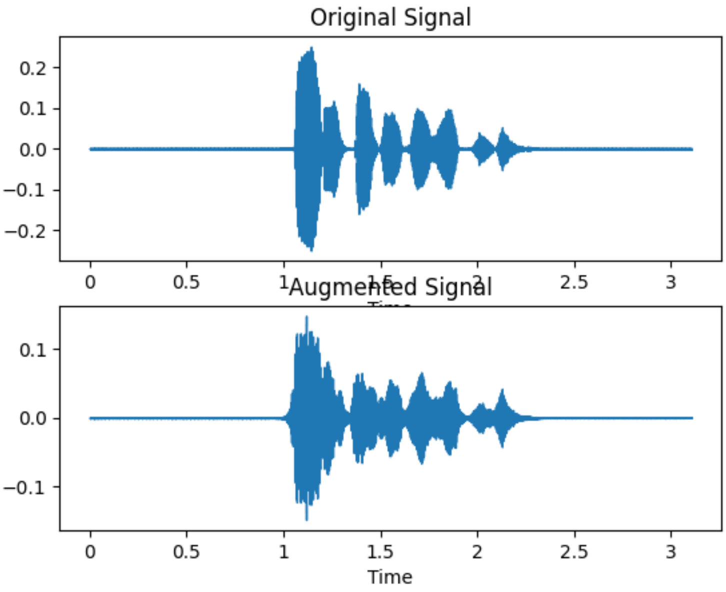
```
path = np.array(dataset_path.Path)[1]
data, sr = librosa.load(path)
augmented_signal = higher_speed(data)
_plot_signal_and_augmented_signal(data, augmented_signal, sr)
```



```
path = np.array(dataset_path.Path)[1]
data, sr = librosa.load(path)
augmented_signal = lower_speed(data)
_plot_signal_and_augmented_signal(data, augmented_signal, sr)
```



```
path = np.array(dataset_path.Path)[1]
data, sr = librosa.load(path)
augmented_signal = pitch(data,22050)
_plot_signal_and_augmented_signal(data, augmented_signal, sr)
```



```
Audio(path, rate=sr)
```

0:00 / 0:03

```
Audio(noise(data), rate=sr)
```

0:00 / 0:03

```
Audio(shift(data), rate=sr)
```

0:00 / 0:03

```
Audio(stretch(data), rate=sr)
```

0:00 / 0:04

```
Audio(lower_speed(data), rate=sr)
```

0:00 / 0:04

```
Audio(higher_speed(data), rate=sr)
```

0:00 / 0:02

```
Audio(pitch(data,22050), rate=sr)
```

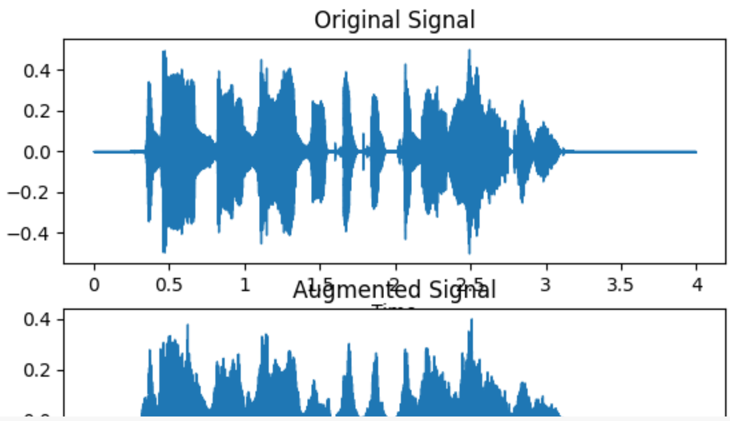
0:00 / 0:03

Feature extraction

```
#sample_rate = 22050

def extract_features(data):

    result = np.array([])

    mfccs = librosa.feature.mfcc(y=data, sr=22050, n_mfcc=58)
    mfccs_processed = np.mean(mfccs.T,axis=0)
    result = np.array(mfccs_processed)

    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=3, offset=0.5)

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

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

    #stretched
    stretch_data = stretch(data)
    res3 = extract_features(stretch_data)
    result = np.vstack((result, res3))

    #shifted
    shift_data = shift(data)
    res4 = extract_features(shift_data)
    result = np.vstack((result, res4))

    #pitched
    pitch_data = pitch(data, sample_rate)
    res5 = extract_features(pitch_data)
    result = np.vstack((result, res5))

    #speed up
    higher_speed_data = higher_speed(data)
    res6 = extract_features(higher_speed_data)
    result = np.vstack((result, res6))

    #speed down
    lower_speed_data = higher_speed(data)
    res7 = extract_features(lower_speed_data)
    result = np.vstack((result, res7))

    return result
```

Checking for error

```
path_new='/content/drive/MyDrive/SUBESCO/M_06_SHUKANTO_S_8_SAD_5.wav'
path = path_new
data, sr = librosa.load(path)
augmented_signal = lower_speed(data)
_plot_signal_and_augmented_signal(data, augmented_signal, sr)
```



```
Audio('/content/drive/MyDrive/SUBESCO/M_06_SHUKANTO_S_8_SAD_5.wav',rate=sr)
```

0:00 / 0:03

Time

Get the Features

```
X, Y = [], []

for path, emotion in zip(dataset_path.Path, dataset_path.Emotions):
    features = get_features(path)
    for feature in features:
        X.append(feature)
        # appending emotion 5 times as we have made 5 augmentation techniques on each audio file.
        Y.append(emotion)
```

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

```
(59269, 59269, (8467,))
```

```
Features = pd.DataFrame(X)
Features['Emotions'] = Y
Features.to_csv('New_Features.csv', index=False)
print(len(Features))
Features.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-d6f28ba551db> in <cell line: 1>()
----> 1 Features = pd.DataFrame(X)
      2 Features['Emotions'] = Y
      3 Features.to_csv('New_Features.csv', index=False)
      4 print(len(Features))
      5 Features.head()

NameError: name 'pd' is not defined
```

SEARCH STACK OVERFLOW

```
Features = pd.read_csv('/content/New_Features.csv')
Features
```

```
<ipython-input-6-4a229d757f58>:1: DtypeWarning: Columns (58) have mixed types. Specify dtype option on import or set low_
  Features = pd.read_csv('/content/New_Features.csv')
                 0           1           2          3          4           5          6           7           8           9  ...        49

     0  -486.931915  115.747589  -25.467674  21.137320  1.408364  -17.748659  -5.206296  -11.816609  -23.193365  -10.483434  ...  2.684004
```

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

```
encoder = OneHotEncoder()
Y_res = encoder.fit_transform(np.array(Y).reshape(-1,1)).toarray()
```

```
x_train, x_test, y_train, y_test = train_test_split(X, Y_res,test_size = 0.2, random_state=42, shuffle=True)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((47415, 58), (47415, 6), (11854, 58), (11854, 6))
 59266  -277.933167   79.442307  -12.162724  23.134661  -0.243249  -22.642027  -8.675693  -17.558695  -17.298103  -10.581200  ...  -0.715106
```

```
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
```

```
((47415, 58, 1), (47415, 6), (11854, 58, 1), (11854, 6))
```

Models

```
!pip3 install -U tensorflow
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.3.3)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.54.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.10)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.12.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.0)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.22.4)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/l
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.12
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tenso
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tens
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->ten
Requirement already satisfied: ml-dtypes>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow)
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow) (1.10
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboar
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from ter
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorboarc
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tens
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oaut
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->t
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorb
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensor
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0-
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization, LSTM, Bidirectional
from keras.utils import np_utils, to_categorical
from keras.callbacks import ModelCheckpoint
```

Model - CNN

```
model = Sequential()
model.add(layers.Conv1D(512, kernel_size=5, strides=1,
                        padding="same", activation="relu",
                        input_shape=(x_train.shape[1], 1)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(512, kernel_size=5, strides=1,
                        padding="same", activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(256, kernel_size=5, strides=1,
                        padding="same", activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(256, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(layers.Conv1D(128, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling1D(pool_size=3, strides = 2, padding = 'same'))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dense(6, activation="softmax"))


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

model.summary()
```

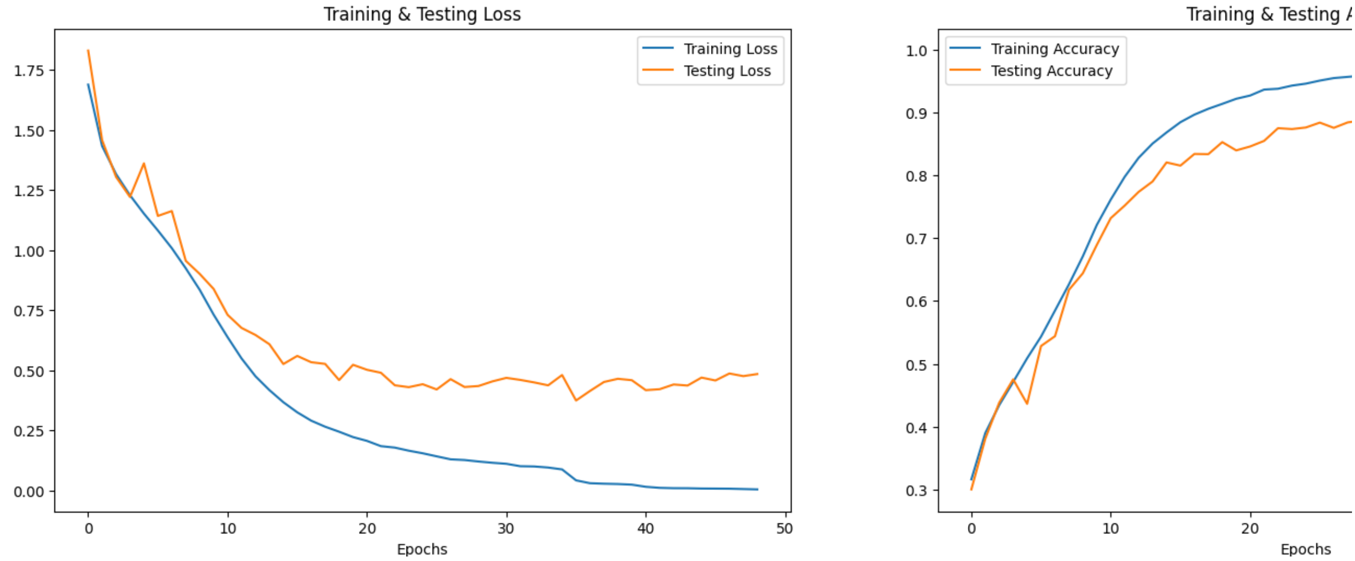
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d_24 (Conv1D)          (None, 58, 512)           3072      
                                                                 
 batch_normalization_28 (Bat  (None, 58, 512)          2048      
 chNormalization)                                                
                                                                 
 max_pooling1d_24 (MaxPoolin  (None, 29, 512)          0         
 g1D)                                                            
                                                                 
 conv1d_25 (Conv1D)          (None, 29, 512)           1311232   
                                                                 
 batch_normalization_29 (Bat  (None, 29, 512)          2048      
 chNormalization)                                                
                                                                 
 max_pooling1d_25 (MaxPoolin  (None, 15, 512)          0         
 g1D)                                                            
                                                                 
 conv1d_26 (Conv1D)          (None, 15, 256)           655616    
                                                                 
 batch_normalization_30 (Bat  (None, 15, 256)          1024      
 chNormalization)                                                
                                                                 
 max_pooling1d_26 (MaxPoolin  (None, 8, 256)           0         
 g1D)                                                            
                                                                 
 conv1d_27 (Conv1D)          (None, 8, 256)            196864    
                                                                 
 batch_normalization_31 (Bat  (None, 8, 256)           1024      
 chNormalization)                                                
                                                                 
 max_pooling1d_27 (MaxPoolin  (None, 4, 256)           0         
 g1D)                                                            
                                                                 
 conv1d_28 (Conv1D)          (None, 4, 128)            98432     
                                                                 
 batch_normalization_32 (Bat  (None, 4, 128)           512       
 chNormalization)                                                
                                                                 
 max_pooling1d_28 (MaxPoolin  (None, 2, 128)           0         
 g1D)                                                            
                                                                 
 conv1d_29 (Conv1D)          (None, 2, 128)            49280     
```

```
batch_normalization_33 (Bat  (None, 2, 128)            512
chNormalization)

max_pooling1d_29 (MaxPoolin  (None, 1, 128)            0
g1D)

flatten_4 (Flatten)         (None, 128)               0

dense_8 (Dense)             (None, 512)               66048

batch_normalization_34 (Bat  (None, 512)               2048
chNormalization)
```

```
rlrp = ReduceLROnPlateau(monitor='val_accuracy',
                                             patience=3,
                                             verbose=1,
                                             factor=0.5,
                                             min_lr=0.00001)
earlystopping = EarlyStopping(monitor ="val_accuracy",
                              mode = 'auto', patience = 5,
                              restore_best_weights = True)

history=model.fit(x_train, y_train, batch_size=64, epochs=70, validation_data=(x_test, y_test), callbacks=[rlrp,earlystopping])
```

```
print("Accuracy of our model on test data : " , round(model.evaluate(x_test,y_test)[1]*100,2) , "%")

epochs = [i for i in range(49)]
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()
```

```
371/371 [==============================] - 4s 9ms/step - loss: 0.4369 - accuracy: 0.9196
Accuracy of our model on test data :  91.96 %
```



Model - RNN(LSTM)

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
timesteps = x_train.shape[1]  # Number of time steps in each sequence
input_dim = x_train.shape[2]  # Number of features in each time step
output_dim = y_train.shape[1]
```

```
model = Sequential()

# Add the first LSTM layer with input shape (timesteps, input_dim)
model.add(LSTM(units=256, return_sequences=True, input_shape=(timesteps, input_dim)))
model.add(Dropout(0.2))

# Add additional LSTM layers
model.add(LSTM(units=256, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=256, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=256, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=256, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=256))
model.add(Dropout(0.2))

# Add a dense layer for output prediction
model.add(Dense(units=output_dim, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm (LSTM)                 (None, 58, 256)           264192    
                                                                 
 dropout (Dropout)           (None, 58, 256)           0         
                                                                 
 lstm_1 (LSTM)               (None, 58, 256)           525312    
                                                                 
 dropout_1 (Dropout)         (None, 58, 256)           0         
                                                                 
 lstm_2 (LSTM)               (None, 58, 256)           525312    
                                                                 
 dropout_2 (Dropout)         (None, 58, 256)           0         
                                                                 
 lstm_3 (LSTM)               (None, 58, 256)           525312    
                                                                 
 dropout_3 (Dropout)         (None, 58, 256)           0         
                                                                 
 lstm_4 (LSTM)               (None, 58, 256)           525312    
                                                                 
 dropout_4 (Dropout)         (None, 58, 256)           0         
                                                                 
 lstm_5 (LSTM)               (None, 256)               525312    
                                                                 
 dropout_5 (Dropout)         (None, 256)               0         
                                                                 
 dense_10 (Dense)            (None, 6)                 1542      
                                                                 
=================================================================
Total params: 2,892,294
Trainable params: 2,892,294
Non-trainable params: 0
_________________________________________________________________
```

```
history=model.fit(x_train, y_train, epochs=100, batch_size=64,validation_data=(x_test, y_test))
```

```
Epoch 1/100
741/741 [==============================] - 1401s 2s/step - loss: 1.7309 - accuracy: 0.2681 - val_loss: 1.7095 - val_accur
Epoch 2/100
741/741 [==============================] - 1375s 2s/step - loss: 1.5954 - accuracy: 0.3305 - val_loss: 1.5066 - val_accur
Epoch 3/100
474/741 [==================>...........] - ETA: 7:48 - loss: 1.4724 - accuracy: 0.3713
```

```
print("Accuracy of our model on test data : " , round(model.evaluate(x_test,y_test)[1]*100,2) , "%")

epochs = [i for i in range(48)]
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()
```

Bi-LSTM

```
from keras.models import Sequential
from keras.layers import LSTM, Bidirectional, Dense
```

```
# Define the number of hidden layers and units per layer
num_hidden_layers = 7
units_per_layer = 256

# Create the model
model = Sequential()

# Add the first Bi-LSTM layer with input_shape
model.add(Bidirectional(LSTM(units_per_layer, return_sequences=True), input_shape=(None, input_dim)))

# Add the remaining hidden layers
for _ in range(num_hidden_layers - 1):
    model.add(Bidirectional(LSTM(units_per_layer, return_sequences=True)))

# Add the output layer
model.add(Dense(output_dim, activation='softmax'))

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

# Print the model summary
model.summary()
```

```
print("Accuracy of our model on test data : " , round(model.evaluate(x_test,y_test)[1]*100,2) , "%")

epochs = [i for i in range(48)]
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()
```