**Title:** Voice Recognition using ML

**Description:**

Voice Recognition using Machine Learning (ML) involves developing a system that accurately converts spoken words into text or commands. This project utilizes machine learning algorithms like CNNs, RNNs, and CRNNs to train models on a diverse dataset of audio recordings. Preprocessing techniques such as MFCC feature extraction and data augmentation enhance model performance. The system is integrated into a user-friendly interface for real-time interaction and deployed on various platforms for widespread usage. Testing evaluates accuracy, response time, and robustness, with future enhancements aimed at improving performance and usability.

**Team Name:** C0deHackz

**Team members name:**

P. Sai Keerthan Reddy

K. Ambica

**Code:**

import os

import sys

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.utils import shuffle

import librosa

import librosa.display

from IPython.display import Audio

import seaborn as sns

sns.set\_style('whitegrid') # or any other Seaborn styleSC

TESS = r"D:\TESS\TESS Toronto emotional speech set data"

RAV = r"D:\RAV\\"

SAVEE = r"D:\SAVEE\\"

CREMA = r"D:\CREMA\\"

import os

SAVEE = r"D:\SAVEE\\"

dir\_list = os.listdir(SAVEE)

# parse the filename to get the emotions

emotion = []

import os

SAVEE = r"D:\SAVEE\\"

dir\_list = os.listdir(SAVEE)

# Now you can continue with your code to parse the filenames and get emotions

dir\_list = os.listdir(SAVEE)

import pandas as pd

# parse the filename to get the emotions

emotion=[]

path = []

for i in dir\_list:

if i[-8:-6]=='\_a':

emotion.append('angry')

elif i[-8:-6]=='\_d':

emotion.append('disgust')

elif i[-8:-6]=='\_f':

emotion.append('fear')

elif i[-8:-6]=='\_h':

emotion.append('happy')

elif i[-8:-6]=='\_n':

emotion.append('neutral')

elif i[-8:-6]=='sa':

emotion.append('sad')

elif i[-8:-6]=='su':

emotion.append('surprise')

# Now check out the label count distribution

SAVEE\_df = pd.DataFrame(emotion, columns = ['labels'])

SAVEE\_df = pd.concat([SAVEE\_df, pd.DataFrame(path, columns = ['path'])], axis = 1)

print('SAVEE dataset')

SAVEE\_df.head()

import os

SAVEE = r"D:\SAVEE\\"

dir\_list = os.listdir(SAVEE)

# parse the filename to get the emotions

emotion=[]

path = []

for i in dir\_list:

if i[-8:-6]=='\_a':

emotion.append('angry')

elif i[-8:-6]=='\_d':

emotion.append('disgust')

elif i[-8:-6]=='\_f':

emotion.append('fear')

elif i[-8:-6]=='\_h':

emotion.append('happy')

elif i[-8:-6]=='\_n':

emotion.append('neutral')

elif i[-8:-6]=='sa':

emotion.append('sad')

elif i[-8:-6]=='su':

emotion.append('surprise')

# Now check out the label count distribution

SAVEE\_df = pd.DataFrame(emotion, columns = ['labels'])

savee\_df = pd.concat([SAVEE\_df, SAVEE\_df], axis = 1)

print('SAVEE dataset')

savee\_df.head()

import os

import pandas as pd

TESS = r"D:\TESS\TESS Toronto emotional speech set data"

path = []

emotion = []

dir\_list = os.listdir(TESS)

for i in dir\_list:

fname = os.listdir(os.path.join(TESS, i))

for f in fname:

if i == 'OAF\_angry' or i == 'YAF\_angry':

emotion.append('angry')

elif i == 'OAF\_disgust' or i == 'YAF\_disgust':

emotion.append('disgust')

elif i == 'OAF\_fear' or i == 'YAF\_fear':

emotion.append('fear')

elif i == 'OAF\_happy' or i == 'YAF\_happy':

emotion.append('happy')

elif i == 'OAF\_neutral' or i == 'YAF\_neutral':

emotion.append('neutral')

elif i == 'OAF\_pleasant\_surprise' or i == 'YAF\_pleasant\_surprise':

emotion.append('surprise')

elif i == 'OAF\_sad' or i == 'YAF\_sad':

emotion.append('sad')

else:

emotion.append('Unknown')

path.append(os.path.join(TESS, i, f))

TESS\_df = pd.DataFrame({'labels': emotion, 'path': path})

tess\_df = pd.concat([TESS\_df, TESS\_df], axis=1)

print('TESS dataset')

print(tess\_df.head())

file = "example-file-name-2024-03-15-17-23-45.wav"

part = file.split('.')[0]

part = part.split("-")

if len(part) > 6:

temp = int(part[6])

if part[2] == '01':

emotion = 'neutral'

else:

# Handle other cases based on part[2]

emotion = 'unknown'

else:

print("Not enough elements in the 'part' list.")

print(emotion)

file = "example-file-name-2024-03-15-17-23-45.wav"

part = file.split('.')[0]

part = part.split("-")

if len(part) > 6:

temp = int(part[6])

if part[2] == '01':

emotion = 'neutral'

else:

# Handle other cases based on part[2]

emotion = 'unknown'

else:

print("Not enough elements in the 'part' list.")

print(emotion)

import os

import pandas as pd

# Replace 'path\_to\_RAV\_directory' with the actual path to your RAV directory

RAV = 'D:\\RAV\\'

males = []

females = []

dir = os.listdir(RAV)

for actor in dir:

files = os.listdir(os.path.join(RAV, actor))

for file in files:

part = file.split('.')[0].split("-")

# Check if the 'part' list has enough elements to proceed

if len(part) >= 7:

gender\_part = part[6].split('\_')

if len(gender\_part) > 1:

gender = gender\_part[1] # Extracting gender information from the file name

else:

gender = 'unknown'

# Check if the 'part' list has enough elements to determine emotion

if len(part) >= 3:

if part[2] == '01':

emotion = 'neutral'

elif part[2] == '02':

emotion = 'calm'

elif part[2] == '03':

emotion = 'happy'

elif part[2] == '04':

emotion = 'sad'

elif part[2] == '05':

emotion = 'angry'

elif part[2] == '06':

emotion = 'fear'

elif part[2] == '07':

emotion = 'disgust'

elif part[2] == '08':

emotion = 'surprise'

else:

emotion = 'unknown'

else:

emotion = 'unknown'

else:

gender = 'unknown'

emotion = 'unknown'

path = os.path.join(RAV, actor, file)

if gender == 'female':

females.append([emotion, path])

else:

males.append([emotion, path])

RavFemales\_df = pd.DataFrame(females, columns=['labels', 'path'])

RavMales\_df = pd.DataFrame(males, columns=['labels', 'path'])

print('RAVDESS datasets')

print('RavFemales\_df:')

print(RavFemales\_df.head())

print('\nRavMales\_df:')

print(RavMales\_df.head())

import os

import pandas as pd

# Replace 'path\_to\_RAV\_directory' with the actual path to your RAV directory

RAV = 'D:\\RAV\\'

males = []

females = []

dir = os.listdir(RAV)

for actor in dir:

files = os.listdir(os.path.join(RAV, actor))

for file in files:

part = file.split('.')[0].split("-")

# Check if the 'part' list has enough elements to proceed

if len(part) >= 7:

gender\_part = part[6].split('\_')

if len(gender\_part) > 1:

gender = gender\_part[1] # Extracting gender information from the file name

else:

gender = 'unknown'

# Check if the 'part' list has enough elements to determine emotion

if len(part) >= 3:

if part[2] == '01':

emotion = 'neutral'

elif part[2] == '02':

emotion = 'calm'

elif part[2] == '03':

emotion = 'happy'

elif part[2] == '04':

emotion = 'sad'

elif part[2] == '05':

emotion = 'angry'

elif part[2] == '06':

emotion = 'fear'

elif part[2] == '07':

emotion = 'disgust'

elif part[2] == '08':

emotion = 'surprise'

else:

emotion = 'unknown'

else:

emotion = 'unknown'

else:

gender = 'unknown'

emotion = 'unknown'

path = os.path.join(RAV, actor, file)

if gender == 'female':

females.append([emotion, path])

else:

males.append([emotion, path])

RavFemales\_df = pd.DataFrame(females, columns=['labels', 'path'])

RavMales\_df = pd.DataFrame(males, columns=['labels', 'path'])

RF\_df = pd.concat([RavFemales\_df, RavFemales\_df], axis=1)

RM\_df = pd.concat([RavMales\_df, RavMales\_df], axis=1)

print('RAVDESS datasets')

print('RavFemales\_df:')

print(RF\_df.head())

print('\nRavMales\_df:')

print(RM\_df.head())

RavMales\_df.head()

RavMales\_df.head()

file = "D:\\RAV\\Actor\_01\\03-01-01-01-01-01-01.wav"

part = file.split('.')[0]

part = part.split("-")

if len(part) > 6:

temp = int(part[6])

if part[2] == '01':

emotion = 'neutral'

else:

# Handle other cases based on part[2]

emotion = 'unknown'

else:

print("Not enough elements in the 'part' list.")

print(emotion)

import os

import pandas as pd

# Replace 'path\_to\_CREMA\_directory' with the actual path to your CREMA directory

CREMA = r"D:\CREMA\\"

files = os.listdir(CREMA)

female = [1002,1003,1004,1006,1007,1008,1009,1010,1012,1013,1018,1020,1021,1024,1025,1028,1029,1030,1037,1043,1046,1047,1049,

1052,1053,1054,1055,1056,1058,1060,1061,1063,1072,1073,1074,1075,1076,1078,1079,1082,1084,1089,1091]

male = [1002,1003,1004,1006,1007,1008,1009,1010,1012,1013,1018,1020,1021,1024,1025,1028,1029,1030,1037,1043,1046,1047,1049,

1052,1053,1054,1055,1056,1058,1060,1061,1063,1072,1073,1074,1075,1076,1078,1079,1082,1084,1089,1091]

males = []

females = []

for file in files:

part = file.split('\_')

if len(part) >= 3: # Check if part has at least 3 elements

if part[2] == 'SAD':

emotion = 'sad'

elif part[2] == 'ANG':

emotion = 'angry'

elif part[2] == 'DIS':

emotion = 'disgust'

elif part[2] == 'FEA':

emotion = 'fear'

elif part[2] == 'HAP':

emotion = 'happy'

elif part[2] == 'NEU':

emotion = 'neutral'

else:

emotion = 'unknown'

if int(part[0]) in female:

path = (CREMA + '/' + file)

females.append([emotion, path])

else:

path = (CREMA + '/' + file)

males.append([emotion, path])

CremaFemales\_df = pd.DataFrame(females)

CremaFemales\_df.columns = ['labels', 'path']

CremaMales\_df = pd.DataFrame(males)

CremaMales\_df.columns = ['labels', 'path']

CF\_df = pd.concat([CremaFemales\_df, CremaFemales\_df], axis=1)

CM\_df = pd.concat([CremaMales\_df,CremaMales\_df], axis=1)

print('CREMA datasets')

CF\_df.head()

CM\_df.head()

CremaMales\_df.head()

# Now lets merge all the dataframe

Males = pd.concat([SAVEE\_df, RavMales\_df, CremaMales\_df], axis = 0)

Males.to\_csv("males\_emotions\_df.csv", index = False)

Females = pd.concat([TESS\_df, RavFemales\_df, CremaFemales\_df], axis = 0)

Females.to\_csv("females\_emotions\_df.csv", index = False)

import pandas as pd

# Assuming you have created and filled data frames like this

SAVEE\_df = pd.DataFrame(data={'emotion': ['happy', 'sad'], 'file': ['file1.wav', 'file2.wav']})

RavMales\_df = pd.DataFrame(data={'emotion': ['angry', 'neutral'], 'file': ['file3.wav', 'file4.wav']})

CremaMales\_df = pd.DataFrame(data={'emotion': ['fear', 'surprise'], 'file': ['file5.wav', 'file6.wav']})

# Concatenating data frames

Males = pd.concat([SAVEE\_df, RavMales\_df, CremaMales\_df], axis=0)

Males.to\_csv("males\_emotions\_df.csv", index=False)

Males = pd.concat([SAVEE\_df, RavMales\_df, CremaMales\_df], axis = 0)

Males.to\_csv("males\_emotions\_df.csv", index = False)

Females = pd.concat([TESS\_df, RavFemales\_df, CremaFemales\_df], axis = 0)

Females.to\_csv("females\_emotions\_df.csv", index = False)

import pandas as pd

# Assuming 'Males' DataFrame has been created and filled with data

# Check if 'labels' and 'path' exist in the DataFrame columns

if 'labels' in Males.columns and 'path' in Males.columns:

# Create a new DataFrame with only 'path' and 'gender' columns

Males\_gender = Males[['path']].copy()

else:

# If 'labels' or 'path' doesn't exist, create an empty DataFrame

Males\_gender = pd.DataFrame(columns=['path'])

# Add a 'gender' column with value 'Male'

Males\_gender['gender'] = "Male"

# Display the first few rows of Males\_gender DataFrame

Males\_gender.head()

Males.head()

Males.count()

Females.head()

Females.head()

Females.count()

import pandas as pd

# Assuming 'Males' DataFrame has been created and filled with data

# Check if 'labels' exists in the DataFrame

if 'labels' in Males.columns:

# Drop the 'labels' column if it exists

Males\_gender = Males.drop('labels', axis=1)

Males\_gender['gender'] = "Male"

Males\_gender.head()

else:

print("The 'labels' column does not exist in the DataFrame.")

import pandas as pd

# Assuming 'Males' DataFrame has been created and filled with data

# Check if 'labels' exists in the DataFrame columns

if 'labels' in Males.columns:

# Drop the 'labels' column along the columns axis (axis=1)

Males\_gender = Males.drop('labels', axis=1)

Males\_gender['gender'] = "Male"

Males\_gender.head()

else:

print("The 'labels' column does not exist in the DataFrame.")

print(Males.columns)

import pandas as pd

# Assuming 'Males' DataFrame has been created and filled with data

# Check column names

print(Males.columns)

# Verify DataFrame content

print(Males.head())

# Check if 'labels' exists in the DataFrame columns

if 'labels' in Males.columns:

# Drop the 'labels' column along the columns axis (axis=1)

Males\_gender = Males.drop('labels', axis=1)

Males\_gender['gender'] = "Male"

Males\_gender.head()

else:

print("The 'labels' column does not exist in the DataFrame.")

import pandas as pd

# Assuming 'Males' DataFrame has been created and filled with data

# Check if 'labels' exists in the DataFrame columns

if 'labels' in Males.columns:

# Drop the 'labels' column along the columns axis (axis=1)

Males\_gender = Males.drop('labels', axis=1)

Males\_gender['gender'] = "Male"

Males\_gender.head()

else:

print("The 'labels' column does not exist in the DataFrame.")

import pandas as pd

# Assuming 'Males' DataFrame has been created and filled with data

# Check if 'labels' exists in the DataFrame columns

if 'labels' in Males.columns:

# Drop the 'labels' column along the columns axis (axis=1)

Males\_gender = Males.drop(columns=['labels'], axis=1, errors='ignore')

Males\_gender['gender'] = "Male"

Males\_gender.head()

else:

print("The 'labels' column does not exist in the DataFrame.")

import pandas as pd

# Assuming 'Males' DataFrame has been created and filled with data

# Check if 'labels' exists in the DataFrame columns

if 'labels' in Males.columns:

# Drop the 'labels' column along the columns axis (axis=1)

Males\_gender = Males.drop(columns=['labels'], axis=1, errors='ignore')

else:

# If 'labels' doesn't exist, just assign Males to Males\_gender

Males\_gender = Males.copy()

# Add a 'gender' column with value 'Male'

Males\_gender['gender'] = "Male"

# Display the first few rows of Males\_gender DataFrame

Males\_gender.head()

Females\_gender=Females.drop('labels', axis=1)

Females\_gender['gender']="Female"

Females\_gender.head()

Gender = pd.concat([Males\_gender,Females\_gender])

Gender = shuffle(Gender,random\_state = 1)

Gender = Gender.reset\_index(drop=True)

Gender.head()

gender\_counts = Gender['gender'].value\_counts()

plt.bar(gender\_counts.index, gender\_counts.values)

plt.title('Count of Females Emotions')

plt.ylabel('Count')

plt.xlabel('Gender')

sns.despine(top=True, right=True, left=False, bottom=False)

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

# Check for duplicate labels in Females DataFrame

duplicate\_labels = Females[Females.duplicated('labels')]['labels']

if not duplicate\_labels.empty:

print("Duplicate labels found:")

print(duplicate\_labels)

# Decide how to handle duplicate labels (e.g., remove duplicates)

Females\_unique = Females.drop\_duplicates('labels')

# Plot the count of female emotions

fig = plt.figure(figsize=(10, 6))

ax = fig.add\_subplot(121)

sns.countplot(x='labels', data=Females\_unique, order=order, ax=ax)

plt.title('Count of Females Emotions', size=16)

plt.ylabel('Count', size=12)

plt.xlabel('Emotions', size=12)

plt.xticks(rotation=45)

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

# Aggregate counts of emotions for females and males

Females\_counts = Females['labels'].value\_counts().reset\_index()

Females\_counts.columns = ['labels', 'count']

Males\_counts = Males['labels'].value\_counts().reset\_index()

Males\_counts.columns = ['labels', 'count']

# Plot the count of emotions for both females and males

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# Plot count of females emotions

sns.barplot(x='labels', y='count', data=Females\_counts, order=order, ax=ax1)

ax1.set\_title('Count of Females Emotions', size=16)

ax1.set\_ylabel('Count', size=12)

ax1.set\_xlabel('Emotions', size=12)

ax1.set\_xticklabels(ax1.get\_xticklabels(), rotation=45)

# Plot count of males emotions

sns.barplot(x='labels', y='count', data=Males\_counts, order=order, ax=ax2)

ax2.set\_title('Count of Males Emotions', size=16)

ax2.set\_ylabel('Count', size=12)

ax2.set\_xlabel('Emotions', size=12)

ax2.set\_xticklabels(ax2.get\_xticklabels(), rotation=45)

plt.tight\_layout()

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming you have already created the DataFrame TESS\_df containing emotion labels

plt.figure(figsize=(8, 6))

plt.title('Count of Emotions', size=16)

sns.countplot(x='labels', data=TESS\_df)

plt.ylabel('Count', size=12)

plt.xlabel('Emotions', size=12)

sns.despine(top=True, right=True, left=False, bottom=False)

plt.xticks(rotation=45) # Rotate x-axis labels for better readability

plt.show()

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import librosa

import librosa.display

# Define the path to your audio file

audio\_path = 'D:/RAV/Actor\_01/03-01-01-01-01-01-01.wav'

# Load the audio data

data, sampling\_rate = librosa.load(audio\_path)

# Extract features related to emotions (this part depends on your specific analysis)

# For example, you can use librosa to extract Mel-frequency cepstral coefficients (MFCCs)

mfccs = librosa.feature.mfcc(y=data, sr=sampling\_rate, n\_mfcc=13)

# Convert the MFCCs to a DataFrame

df = pd.DataFrame(mfccs.T, columns=[f'MFCC{i}' for i in range(1, 14)])

# Add an 'Emotions' column to the DataFrame (just for illustration)

# You need to replace this with your actual emotion labels

df['Emotions'] = ['happy'] \* len(df) # Example: Assigning 'happy' for all samples

# Assuming df is your DataFrame containing the data and Emotions column

plt.title('Count of Emotions', size=16)

sns.countplot(df['Emotions'])

plt.ylabel('Count', size=12)

plt.xlabel('Emotions', size=12)

plt.show()

import matplotlib.pyplot as plt

import librosa

def create\_waveshow(data, sr, e):

plt.figure(figsize=(10, 3))

plt.title(f'Waveshow for audio with {e} emotion', size=15)

# Calculate time axis in seconds

time = librosa.times\_like(data, sr=sr)

# Plot the waveform using matplotlib

plt.plot(time, data)

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

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()

!pip install librosa

pip install setuptools

import librosa

import librosa.display

import matplotlib.pyplot as plt

from IPython.display import Audio

# Define the path to your audio file

path = 'D:/RAV/Actor\_01/03-01-01-01-01-01-01.wav'

emotion = 'Angry'

data, sampling\_rate = librosa.load(path)

create\_waveshow(data, sampling\_rate, emotion)

create\_spectrogram(data, sampling\_rate, emotion)

Audio(path)

emotion='Sing Very Angry'

path = 'D:/RAV/Actor\_01/03-01-05-02-01-01-01.wav'

data, sampling\_rate = librosa.load(path)

create\_waveshow(data, sampling\_rate, emotion)

create\_spectrogram(data, sampling\_rate, emotion)

Audio(path)

emotion='Sing Angry'

path = 'D:/RAV\Actor\_01/03-01-05-02-01-02-01.wav'

data, sampling\_rate = librosa.load(path)

create\_waveshow(data, sampling\_rate, emotion)

create\_spectrogram(data, sampling\_rate, emotion)

Audio(path)

def noise(data):

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)

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=0.8):

return librosa.effects.pitch\_shift(data, sampling\_rate, pitch\_factor)

def higher\_speed(data, speed\_factor = 1.25):

return librosa.effects.time\_stretch(data, speed\_factor)

def lower\_speed(data, speed\_factor = 0.75):

return librosa.effects.time\_stretch(data, speed\_factor)

# taking any example and checking for techniques.

path = path = 'D:/RAV/Actor\_01/03-01-05-01-01-01-01.wav'

data, sample\_rate = librosa.load(path)

import matplotlib.pyplot as plt

import librosa

from IPython.display import Audio

# Load audio data and sample rate

y, sr = librosa.load('D:/RAV/Actor\_01/03-01-05-01-01-01-01.wav')

# Plot the waveform

plt.figure(figsize=(10, 3))

plt.plot(y)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Waveform')

plt.show()

# Generate noise with the same length as y

noise\_data = np.random.normal(0, 0.02, len(y))

# Plot the noisy waveform

plt.figure(figsize=(10, 3))

plt.plot(noise\_data)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Noisy Waveform')

plt.show()

# Add noise to the original audio

noisy\_audio = y + noise\_data

# Display the noisy audio waveform

plt.figure(figsize=(10, 3))

plt.plot(noisy\_audio)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Noisy Audio Waveform')

plt.show()

# Play the noisy audio

Audio(data=noisy\_audio, rate=sr)

import matplotlib.pyplot as plt

import librosa

from IPython.display import Audio

# Load audio data and sample rate

y, sr = librosa.load('D:/RAV/Actor\_01/03-01-05-01-01-01-01.wav')

# Plot the waveform

plt.figure(figsize=(10, 3))

plt.plot(y)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Waveform')

plt.show()

# Generate noise with the same length as y

strech\_data = np.random.normal(0, 0.02, len(y))

# Plot the noisy waveform

plt.figure(figsize=(10, 3))

plt.plot(strech\_data)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('stretch Waveform')

plt.show()

# Add noise to the original audio

noisy\_audio = y + noise\_data

# Display the noisy audio waveform

plt.figure(figsize=(10, 3))

plt.plot(stretch\_audio)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('stretch Audio Waveform')

plt.show()

# Play the noisy audio

Audio(data=stretch\_audio, rate=sr)

import matplotlib.pyplot as plt

import librosa

from IPython.display import Audio

# Load audio data and sample rate

y, sr = librosa.load('D:/RAV/Actor\_01/03-01-05-01-01-01-01.wav')

# Plot the waveform

plt.figure(figsize=(10, 3))

plt.plot(y)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Waveform')

plt.show()

# Generate noise with the same length as y

shift\_data = np.random.normal(0, 0.02, len(y))

# Plot the noisy waveform

plt.figure(figsize=(10, 3))

plt.plot(shift\_data)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('shift Waveform')

plt.show()

shift\_audio = y + shift\_data

plt.figure(figsize=(10, 3))

plt.plot(shift\_audio)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('shift Audio Waveform')

plt.show()

Audio(data=shift\_audio, rate=sr)

import matplotlib.pyplot as plt

import librosa

from IPython.display import Audio

# Load audio data and sample rate

y, sr = librosa.load('D:/RAV/Actor\_01/03-01-05-01-01-01-01.wav')

# Plot the original waveform

plt.figure(figsize=(10, 3))

plt.plot(y)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Original Waveform')

plt.show()

# Change the speed of the audio (higher speed)

higher\_speed\_audio = librosa.effects.time\_stretch(y, rate=1.5)

# Plot the noisy waveform

plt.figure(figsize=(10, 3))

plt.plot(higher\_speed\_data)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('higher\_speed Waveform')

plt.show()

# Plot the higher speed waveform

plt.figure(figsize=(10, 3))

plt.plot(higher\_speed\_audio)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Higher Speed Waveform')

plt.show()

# Play the higher speed audio

Audio(data=higher\_speed\_audio, rate=sr)

import matplotlib.pyplot as plt

import librosa

from IPython.display import Audio

# Load audio data and sample rate

y, sr = librosa.load('D:/RAV/Actor\_01/03-01-05-01-01-01-01.wav')

# Plot the original waveform

plt.figure(figsize=(10, 3))

plt.plot(y)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Original Waveform')

plt.show()

# Slow down the audio by a factor of 2

slow\_audio = librosa.effects.time\_stretch(y, rate=0.5)

# Plot the slowed down waveform

plt.figure(figsize=(10, 3))

plt.plot(slow\_audio)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Slowed Down Waveform')

plt.show()

# Display the slowed down audio waveform

plt.figure(figsize=(10, 3))

plt.plot(slow\_audio)

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.title('Slowed Down Audio Waveform')

plt.show()

# Play the slowed down audio

Audio(data=slow\_audio, rate=sr)

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

# creating Dataframe using all the 4 dataframes we created so far.

data\_path = pd.concat([tess\_df,savee\_df], axis = 1)

data\_path.to\_csv("data\_path.csv",index=False)

data\_path.head()

#trail for csv file

# Reset index for tess\_df and savee\_df

tess\_df.reset\_index(drop=True, inplace=True)

savee\_df.reset\_index(drop=True, inplace=True)

RM\_df.reset\_index(drop=True, inplace=True)

RF\_df.reset\_index(drop=True, inplace=True)

CF\_df.reset\_index(drop=True, inplace=True)

CM\_df.reset\_index(drop=True, inplace=True)

# Concatenate the DataFrames

data\_path = pd.concat([tess\_df, savee\_df,RM\_df,RF\_df,CM\_df,CF\_df], axis=1)

# Save the concatenated DataFrame to a CSV file

data\_path.to\_csv("data\_path.csv", index=False)

# Display the first few rows of the concatenated DataFrame

data\_path.head()

data\_path = pd.concat([tess\_df, savee\_df,RM\_df,RF\_df,CM\_df,CF\_df], axis = 1)

data\_path.to\_csv("data\_path.csv",index=False)

data\_path.head()

X, Y = [], []

for path, emotion in zip(data\_path.path, data\_path.labels):

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)

import pandas as pd

def get\_features(audio\_path):

# Placeholder for feature extraction logic

# You should implement your actual feature extraction process here

features = [1, 2, 3] # Placeholder features

return features

# Assuming 'data\_path' is your DataFrame with columns 'path' and 'labels'

data\_path = pd.DataFrame({'path': ["D:\Actor\_03\03-01-01-01-01-01-03.wav", "D:\Actor\_03\03-01-01-01-01-02-03.wav"],

'labels': ['happy', 'sad']})

X, Y = [], []

for path, emotion in zip(data\_path['path'], data\_path['labels']):

# Replace 'get\_features' with your actual feature extraction function

feature = get\_features(path)

for ele in feature:

X.append(ele)

Y.append(emotion)

# Now you can use the feature matrix X and labels Y for further processing or modeling

print("Features:", X)

print("Labels:", Y)

X, Y = [], []

for path, emotion in zip(data\_path.path, data\_path.labels):

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)

import pandas as pd

# Read the CSV file into a DataFrame

data\_path = pd.read\_csv("features.csv")

# Output the column names to identify the correct column containing file paths

print(data\_path.columns)

# Get the shape of the 'filename' column in the DataFrame

path\_column\_shape = data\_path['labels'].shape

len(X), len(Y), data\_path.labels.shape

import pandas as pd

# Create a DataFrame with file paths and labels

data\_path = pd.DataFrame({

'path': ['D:\\dataset\\features.csv', 'D:\\RAV'],

'labels': ['happy', 'sad']

})

# Display the DataFrame and its shape

print("Length of X:", len(X))

print("Length of Y:", len(Y))

print("Shape of data\_path:", data\_path.shape)

data\_path

len(X), len(Y), data\_path['path'].shape

import pandas as pd

# Assuming 'data\_path' is your DataFrame with columns 'path' and 'labels'

data\_path = pd.DataFrame({'path': ["D:\dataset\features.csv","D:\RAV"],

'labels': ['happy', 'sad']})

X, Y = [], []

for path, emotion in zip(data\_path['path'], data\_path['labels']):

# Use the get\_features function to extract features from audio files

feature = get\_features(path)

for ele in feature:

X.append(ele)

Y.append(emotion)

# Check the length of X, Y, and the shape of data\_path

print("Length of X:", len(X))

print("Length of Y:", len(Y))

print("Shape of data\_path:", data\_path.shape)

Features = pd.DataFrame(X)

Features['labels'] = Y

Features.to\_csv('features.csv', index=False)

Features.head(20)

DATA\_FRAMES = True

features\_path = "D:\dataset\features.csv"

import pandas as pd

# Assuming 'features\_path' is a DataFrame with a column named 'labels'

features\_path = pd.DataFrame({'path': ['D:\dataset\features.csv'],

'labels': ['label1']})

features\_path.head()

DATA\_FRAMES = True

fem\_path = "D:\dataset\Female\_features.csv"

mal\_path = "D:\dataset\Male\_features.csv"

def extract\_features(data):

result = np.array([])

#mfccs = librosa.feature.mfcc(y=data, sr=22050, n\_mfcc=42) #42 mfcc so we get frames of ~60 ms

mfccs = librosa.feature.mfcc(y=data, sr=22050, n\_mfcc=34)

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, res\_type='kaiser\_fast')

#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

if not DATA\_FRAMES:

female\_X, female\_Y = [], []

for path, emotion in zip(Females.path, Females.labels):

features = get\_features(path)

#adding augmentation, get\_features return a multi dimensional array (for each augmentation), so we have to use a loop to fill the df

for elem in features:

female\_X.append(elem)

female\_Y.append(emotion)

male\_X, male\_Y = [], []

for path, emotion in zip(Males.path, Males.labels):

features = get\_features(path)

for elem in features:

male\_X.append(elem)

male\_Y.append(emotion)

print(f'Check shapes:\nFemale features: {len(female\_X)}, labels: {len(female\_Y)}\nMale features: {len(male\_X)}, labels: {len(male\_Y)}')

len(X), len(Y), data\_path.path.shape

def setup\_dataframe(gender, features, labels):

df = pd.DataFrame(features)

df['labels'] = labels

df.to\_csv(f'{gender}\_features.csv', index=False)

print(f'{gender} dataframe')

df.sample(frac=1).head()

return df

import pandas as pd

if not DATA\_FRAMES:

females\_Features = setup\_dataframe('Female', female\_X, female\_Y)

else:

females\_Features = pd.read\_csv(fem\_path)

if not DATA\_FRAMES:

Males\_Features = setup\_dataframe('Male', male\_X, male\_Y)

else:

Males\_Features = pd.read\_csv(mal\_path)

Males\_gender=Males\_Features.drop('labels', axis=1)

Males\_gender['gender']="Male"

Males\_gender.head()

Females\_gender=females\_Features.drop('labels', axis=1)

Females\_gender['gender']="Female"

Females\_gender.head()

from sklearn.utils import shuffle

Gender = pd.concat([Males\_gender,Females\_gender])

Gender = shuffle(Gender,random\_state = 1)

Gender = Gender.reset\_index(drop=True)

Gender.head()

import pandas as pd

# Assuming Males\_Features is a DataFrame or Series

Males\_Features = pd.DataFrame({'labels': ['Happy', 'Sad', 'Neutral'], 'feature1': [0.5, 0.3, 0.2]})

# Now you can use Males\_Features in your code

Males\_gender = Males\_Features.drop('labels', axis=1)

Males\_gender['gender'] = "Male"

Males\_gender.head()

Males\_gender=Males\_Features.drop('labels', axis=1)

Males\_gender['gender']="Male"

Males\_gender.head()

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.metrics import confusion\_matrix, classification\_report

from sklearn.model\_selection import train\_test\_split

Gender\_X = Gender.iloc[:,:-1].values

Gender\_Y = Gender['gender'].values

female\_X = females\_Features.iloc[: ,:-1].values

female\_Y = females\_Features['labels'].values

male\_X = Males\_Features.iloc[: ,:-1].values

male\_Y = Males\_Features['labels'].values

import numpy as np # Add this line to import NumPy

encoder1 = OneHotEncoder()

Gender\_Y = encoder1.fit\_transform(np.array(Gender\_Y).reshape(-1,1)).toarray()

encoder = OneHotEncoder()

female\_Y = encoder.fit\_transform(np.array(female\_Y).reshape(-1,1)).toarray()

male\_Y = encoder.fit\_transform(np.array(male\_Y).reshape(-1,1)).toarray()

print(Gender\_Y)

Gender\_Xtrain, Gender\_Xtest, Gender\_Ytrain, Gender\_Ytest = train\_test\_split(Gender\_X, Gender\_Y, random\_state=0, test\_size=0.20, shuffle=True)

Gender\_Xtrain.shape, Gender\_Xtest.shape, Gender\_Ytrain.shape, Gender\_Ytest.shape

x\_trainF, x\_testF, y\_trainF, y\_testF = train\_test\_split(female\_X, female\_Y, random\_state=0, test\_size=0.20, shuffle=True)

x\_trainF.shape, y\_trainF.shape, x\_testF.shape, y\_testF.shape

x\_trainM, x\_testM, y\_trainM, y\_testM = train\_test\_split(male\_X, male\_Y, random\_state=0, test\_size=0.20, shuffle=True)

x\_trainM.shape, y\_trainM.shape, x\_testM.shape, y\_testM.shape

scaler = StandardScaler()

Gender\_Xtrain = scaler.fit\_transform(Gender\_Xtrain)

Gender\_Xtest = scaler.transform(Gender\_Xtest)

x\_trainF = scaler.fit\_transform(x\_trainF)

x\_testF = scaler.transform(x\_testF)

x\_trainM = scaler.fit\_transform(x\_trainM)

x\_testM = scaler.transform(x\_testM)

Gender\_Xtrain\_ml = Gender\_Xtrain

Gender\_Xtest\_ml= Gender\_Xtest

x\_trainF\_ml = x\_trainF

x\_testF\_ml = x\_testF

x\_trainM\_ml = x\_trainM

x\_testM\_ml = x\_testM

Gender\_Xtrain = np.expand\_dims(Gender\_Xtrain, axis=2)

Gender\_Xtest = np.expand\_dims(Gender\_Xtest, axis=2)

Gender\_Xtrain.shape, Gender\_Xtest.shape, Gender\_Ytrain.shape, Gender\_Ytest.shape

x\_trainF = np.expand\_dims(x\_trainF, axis=2)

x\_testF = np.expand\_dims(x\_testF, axis=2)

x\_trainF.shape, y\_trainF.shape, x\_testF.shape, y\_testF.shape

pip install tensorflow

X = Features.iloc[: ,:-1].values

Y = Features['labels'].values

pip install tensorflow

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Dropout, Flatten, Dense

# Initialize the model

model = Sequential()

# Add Conv1D layer with 256 filters, kernel size 5, relu activation, and input shape

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'))

# Add more Conv1D layers with decreasing number of filters

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'))

# Flatten the output and add Dense layers

model.add(Flatten())

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

model.add(Dropout(0.3))

# Output layer with softmax activation for multiclass classification

model.add(Dense(units=8, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Display model summary

model.summary()

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Dropout, Flatten, Dense

from tensorflow.keras.callbacks import ReduceLROnPlateau

# Initialize the model

model = Sequential()

# Add Conv1D layer with 256 filters, kernel size 5, relu activation, and input shape

model.add(Conv1D(256, kernel\_size=5, strides=1, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=5, strides=2, padding='same'))

# Add more Conv1D layers with decreasing number of filters

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'))

# Flatten the output and add Dense layers

model.add(Flatten())

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

model.add(Dropout(0.3))

# Output layer with softmax activation for multiclass classification

model.add(Dense(units=8, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Display model summary

model.summary()

# Fit the model with your data

history = model.fit(x\_train, y\_train\_encoded, batch\_size=64, epochs=50, validation\_data=(x\_test, y\_test\_encoded), callbacks=[rlrp])

import os

SAVEE = r"D:\SAVEE\\"

dir\_list = os.listdir(SAVEE)

# Now you can continue with your code to parse the filenames and get emotions