ser

November 18, 2024

[ ]:
```
import pandas as pd
import numpy as np
import os
import sys
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
import IPython.display as ipd
from IPython.display import Audio
import keras
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM,BatchNormalization , GRU
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
from keras.models import Model
from keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import SGD

import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
warnings.filterwarnings("ignore", category=DeprecationWarning)
import tensorflow as tf
print ("Done")
```

```
Done
```

[ ]:
```
!apt-get update
!apt-get install -y libsndfile1
```

```
[ ]: ravdess = "/kaggle/input/ravdess-emotional-speech-audio/
     ↪audio_speech_actors_01-24/"
     ravdess_directory_list = os.listdir(ravdess)
     print(ravdess_directory_list)
```

```
[ ]: 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 preprocessing

```
[ ]: file_emotion = []
     file_path = []
     for i in ravdess_directory_list:
         actor = os.listdir(ravdess + i)
         for f in actor:
             part = f.split('.')[0].split('-')
             file_emotion.append(int(part[2]))
             file_path.append(ravdess + i + '/' + f)
```

```
[ ]: print(actor[0])
     print(part[0])
     print(file_path[0])
     print(int(part[2]))
     print(f)
```

```
[ ]: emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])
     path_df = pd.DataFrame(file_path, columns=['Path'])
     ravdess_df = pd.concat([emotion_df, path_df], axis=1)
     ravdess_df.Emotions.replace({1:'neutral', 2:'neutral', 3:'happy', 4:'sad', 5:
     ↪'angry', 6:'fear', 7:'disgust', 8:'surprise'},inplace=True)
     print(ravdess_df.head())
     print("______________________________________________")
     print(ravdess_df.tail())
     print("_______________________________________________")
     print(ravdess_df.Emotions.value_counts())
```

CREMA dataset

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

     file_emotion = []
     file_path = []

     for file in crema_directory_list:
```

```
    file_path.append(Crema + file)
    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')

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

TESS dataset

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

file_emotion = []
file_path = []

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

emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])
path_df = pd.DataFrame(file_path, columns=['Path'])
Tess_df = pd.concat([emotion_df, path_df], axis=1)
Tess_df.head()
print(Tess_df.Emotions.value_counts())
```

SAVEE Dataset

[ ]:
```
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')

emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])
path_df = pd.DataFrame(file_path, columns=['Path'])
Savee_df = pd.concat([emotion_df, path_df], axis=1)
Savee_df.head()
print(Savee_df.Emotions.value_counts())
```

Integration

[ ]:
```
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()
```

[ ]:
```
print(data_path.Emotions.value_counts())
```

Data Visualisation and Exploration

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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()
```

[ ]:
```
data,sr = librosa.load(file_path[0])
sr
```

[ ]:
```
ipd.Audio(data,rate=sr)
```

[ ]:
```
plt.figure(figsize=(10, 5))
spectrogram = librosa.feature.melspectrogram(y=data, sr=sr,␣
 ↪n_mels=128,fmax=8000)
log_spectrogram = librosa.power_to_db(spectrogram)
librosa.display.specshow(log_spectrogram, y_axis='mel', sr=sr, x_axis='time');
plt.title('Mel Spectrogram ')
plt.colorbar(format='%+2.0f dB')
```

[ ]:
```
mfcc = librosa.feature.mfcc(y=data, sr=sr, n_mfcc=30)

plt.figure(figsize=(16, 10))
plt.subplot(3,1,1)
librosa.display.specshow(mfcc, x_axis='time')
plt.ylabel('MFCC')
plt.colorbar()

ipd.Audio(data,rate=sr)
```

Data augmentation

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

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

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

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

[ ]:
```
# NORMAL AUDIO

import librosa.display
plt.figure(figsize=(12, 5))
librosa.display.waveshow(y=data, sr=sr)
```

```
ipd.Audio(data,rate=sr)
```

[ ]:
```
# AUDIO WITH NOISE
x = noise(data)
plt.figure(figsize=(12,5))
librosa.display.waveshow(y=x, sr=sr)
ipd.Audio(x, rate=sr)
```

[ ]:
```
# STRETCHED AUDIO
x = stretch(data)
plt.figure(figsize=(12, 5))
librosa.display.waveshow(y=x, sr=sr)
ipd.Audio(x, rate=sr)
```

[ ]:
```
# SHIFTED AUDIO
x = shift(data)
plt.figure(figsize=(12,5))
librosa.display.waveshow(y=x, sr=sr)
ipd.Audio(x, rate=sr)
```

[ ]:
```
# AUDIO WITH PITCH
x = pitch(data, sr)
plt.figure(figsize=(12, 5))
librosa.display.waveshow(y=x, sr=sr)
ipd.Audio(x, rate=sr)
```

Feature extraction

[ ]:
```
def zcr(data,frame_length,hop_length):
    zcr=librosa.feature.
 ↪zero_crossing_rate(data,frame_length=frame_length,hop_length=hop_length)
    return np.squeeze(zcr)
def rmse(data,frame_length=2048,hop_length=512):
    rmse=librosa.feature.
 ↪rms(data,frame_length=frame_length,hop_length=hop_length)
    return np.squeeze(rmse)
def mfcc(data,sr,frame_length=2048,hop_length=512,flatten:bool=True):
    mfcc=librosa.feature.mfcc(data,sr=sr)
    return np.squeeze(mfcc.T)if not flatten else np.ravel(mfcc.T)

def extract_features(data,sr=22050,frame_length=2048,hop_length=512):
    result=np.array([])

    result=np.hstack((result,
                      zcr(data,frame_length,hop_length),
                      rmse(data,frame_length,hop_length),
                      mfcc(data,sr,frame_length,hop_length)
```

```
                    ))
    return result

def get_features(path,duration=2.5, offset=0.6):
    data,sr=librosa.load(path,duration=duration,offset=offset)
    aud=extract_features(data)
    audio=np.array(aud)

    noised_audio=noise(data)
    aud2=extract_features(noised_audio)
    audio=np.vstack((audio,aud2))

    pitched_audio=pitch(data,sr)
    aud3=extract_features(pitched_audio)
    audio=np.vstack((audio,aud3))

    pitched_audio1=pitch(data,sr)
    pitched_noised_audio=noise(pitched_audio1)
    aud4=extract_features(pitched_noised_audio)
    audio=np.vstack((audio,aud4))

    return audio
```

[ ]:
```
import multiprocessing as mp
print("Number of processors: ", mp.cpu_count())
```

Normal way to get features

[ ]:
```
import timeit
from tqdm import tqdm
start = timeit.default_timer()
X,Y=[],[]
for path,emotion,index in tqdm (zip(data_path.Path,data_path.
 ↪Emotions,range(data_path.Path.shape[0]))):
    features=get_features(path)
    if index%500==0:
        print(f'{index} audio has been processed')
    for i in features:
        X.append(i)
        Y.append(emotion)
print('Done')
stop = timeit.default_timer()

print('Time: ', stop - start)
```

[ ]:
```
"""from joblib import Parallel, delayed
import timeit
```

```
start = timeit.default_timer()
# Define a function to get features for a single audio file
def process_feature(path, emotion):
    features = get_features(path)
    X = []
    Y = []
    for ele in features:
        X.append(ele)
        # appending emotion 3 times as we have made 3 augmentation techniques␣
↪on each audio file.
        Y.append(emotion)
    return X, Y

paths = data_path.Path
emotions = data_path.Emotions

# Run the loop in parallel
results = Parallel(n_jobs=-1)(delayed(process_feature)(path, emotion) for␣
↪(path, emotion) in zip(paths, emotions))

# Collect the results
X = []
Y = []
for result in results:
    x, y = result
    X.extend(x)
    Y.extend(y)


stop = timeit.default_timer()

print('Time: ', stop - start)    """
```

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

Saving features

[ ]:
```
Emotions = pd.DataFrame(X)
Emotions['Emotions'] = Y
Emotions.to_csv('emotion.csv', index=False)
Emotions.head()
```

[ ]:
```
Emotions = pd.read_csv('./emotion.csv')
Emotions.head()
```

[ ]:
```
print(Emotions.isna().any())
```

[ ]:
```
Emotions=Emotions.fillna(0)
print(Emotions.isna().any())
Emotions.shape
```

[ ]:
```
np.sum(Emotions.isna())
```

Data preparation

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

[ ]:
```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
encoder = OneHotEncoder()
Y = encoder.fit_transform(np.array(Y).reshape(-1,1)).toarray()
```

[ ]:
```
print(Y.shape)
X.shape
```

[ ]:
```
from sklearn.model_selection import train_test_split

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

[ ]:
```
X_train = x_train.reshape(x_train.shape[0] , x_train.shape[1] , 1)
X_test = x_test.reshape(x_test.shape[0] , x_test.shape[1] , 1)
```

[ ]:
```
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
```

[ ]:
```
import keras
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM,BatchNormalization , GRU
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
from keras.models import Model
from keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import SGD
```

Applying early stopping for all models

[ ]:
```
from keras.callbacks import ModelCheckpoint, EarlyStopping,ReduceLROnPlateau
model_checkpoint = ModelCheckpoint('best_model1_weights.h5',␣
↪monitor='val_accuracy', save_best_only=True)
```

[ ]:
```
early_stop=EarlyStopping(monitor='val_acc',mode='auto',patience=5,restore_best_weights=True)
lr_reduction=ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.
↪5,min_lr=0.00001)
```

LSTM Model

[ ]:
```
"""model01=Sequential()
model01.add(LSTM(128,return_sequences=True,input_shape=(x_train.shape[1],1)))
model01.add(Dropout(0.2))
model01.add(LSTM(128,return_sequences=True))
#model01.add(Dropout(0.2))
model01.add(LSTM(128,return_sequences=True))
#model01.add(Dropout(0.2))
model01.add(LSTM(128,return_sequences=True))
#model01.add(Dropout(0.2))
model01.add(LSTM(128,return_sequences=True))
#model01.add(Dropout(0.2))
model01.add(LSTM(128,return_sequences=True))
#model01.add(Dropout(0.3))
model01.add(LSTM(128))
#model01.add(Dropout(0.3))
model01.add(Dense(7,activation = 'softmax'))
model01.
↪compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model01.summary()"""
```

[ ]:
```
"""hist=model01.fit(X_train, y_train,
            epochs=20,
            validation_data=(X_test, y_test),batch_size=64,
            verbose=1)"""
```

[ ]:
```
"""print("Accuracy of our model on test data : " , model01.
↪evaluate(X_test,y_test)[1]*100 , "%")
epochs = [i for i in range(20)]
fig , ax = plt.subplots(1,2)
train_acc = hist.history['accuracy']
train_loss = hist.history['loss']
test_acc = hist.history['val_accuracy']
test_loss = hist.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()"""
```

CNN model

[ ]:
```
x_traincnn =np.expand_dims(x_train, axis=2)
x_testcnn= np.expand_dims(x_test, axis=2)
x_traincnn.shape, y_train.shape, x_testcnn.shape, y_test.shape
```

[ ]:
```
import tensorflow.keras.layers as L

model = tf.keras.Sequential([
    L.Conv1D(512,kernel_size=5, strides=1,padding='same',␣
 ↪activation='relu',input_shape=(X_train.shape[1],1)),
    L.BatchNormalization(),
    L.MaxPool1D(pool_size=5,strides=2,padding='same'),

    L.Conv1D(512,kernel_size=5,strides=1,padding='same',activation='relu'),
    L.BatchNormalization(),
    L.MaxPool1D(pool_size=5,strides=2,padding='same'),
    Dropout(0.2),

    L.Conv1D(256,kernel_size=5,strides=1,padding='same',activation='relu'),
    L.BatchNormalization(),
    L.MaxPool1D(pool_size=5,strides=2,padding='same'),

    L.Conv1D(256,kernel_size=3,strides=1,padding='same',activation='relu'),
    L.BatchNormalization(),
    L.MaxPool1D(pool_size=5,strides=2,padding='same'),
    Dropout(0.2),

    L.Conv1D(128,kernel_size=3,strides=1,padding='same',activation='relu'),
    L.BatchNormalization(),
    L.MaxPool1D(pool_size=3,strides=2,padding='same'),
    Dropout(0.2),

    L.Flatten(),
    L.Dense(512,activation='relu'),
    L.BatchNormalization(),
```

```
    L.Dense(7,activation='softmax')
])
model.
 ↪compile(optimizer='adam',loss='categorical_crossentropy',metrics='accuracy')
model.summary()
```

[ ]:
```
history=model.fit(x_traincnn, y_train, epochs=50, validation_data=(x_testcnn,␣
 ↪y_test), batch_size=64,callbacks=[early_stop,lr_reduction,model_checkpoint])
```

[ ]:
```
print("Accuracy of our model on test data : " , model.
 ↪evaluate(x_testcnn,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()
```

[ ]:
```
# predicting on test data.
pred_test0 = model.predict(x_testcnn)
y_pred0 = encoder.inverse_transform(pred_test0)
y_test0 = encoder.inverse_transform(y_test)

# Check for random predictions
df0 = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
df0['Predicted Labels'] = y_pred0.flatten()
df0['Actual Labels'] = y_test0.flatten()

df0.head(10)
```

[ ]:
```
df0
```

CLSTM Model

```
[ ]: """model000 = Sequential()
model000.add(Conv1D(1024, kernel_size=5, strides=1, padding='same',␣
 ↪activation='relu', input_shape=(X.shape[1], 1)))
model000.add(MaxPooling1D(pool_size=2, strides = 2, padding = 'same'))
model000.add(BatchNormalization())
model000.add(Dropout(0.3))


model000.add(Conv1D(512, kernel_size=5, strides=1, padding='same',␣
 ↪activation='relu'))
model000.add(MaxPooling1D(pool_size=2, strides = 2, padding = 'same'))
model000.add(BatchNormalization())
model000.add(Dropout(0.3))

model000.add(Conv1D(256, kernel_size=5, strides=1, padding='same',␣
 ↪activation='relu'))
model000.add(MaxPooling1D(pool_size=2, strides = 2, padding = 'same'))
model000.add(BatchNormalization())
model000.add(Dropout(0.3))

model000.add(LSTM(128, return_sequences=True))
model000.add(Dropout(0.3))

model000.add(LSTM(128, return_sequences=True))
model000.add(Dropout(0.3))
model000.add(LSTM(128))
model000.add(Dropout(0.3))

model000.add(Dense(128, activation='relu'))
#model000.add(Dropout(0.3))

model000.add(Dense(64, activation='relu'))
#model000.add(Dropout(0.3))

model000.add(Dense(32, activation='relu'))
#model000.add(Dropout(0.3))

model000.add(Dense(7, activation='softmax'))



model000.summary()"""
```

```
[ ]: """from keras.utils.vis_utils import plot_model
```

```
plot_model( model000, show_shapes=True, show_layer_names=True,␣
↪to_file='model000.png')"""
```

[ ]:
```
"""model000.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])"""
```

[ ]:
```
"""hist1=model000.fit(x_traincnn, y_train, batch_size=64, epochs=40,␣
↪validation_data=(x_testcnn, y_test))"""
```

[ ]:
```
"""print("Accuracy of our model on test data : " , model000.
↪evaluate(x_testcnn,y_test)[1]*100 , "%")
epochs = [i for i in range(40)]
fig , ax = plt.subplots(1,2)
train_acc = hist1.history['accuracy']
train_loss = hist1.history['loss']
test_acc = hist1.history['val_accuracy']
test_loss = hist1.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()"""
```

[ ]:
```
# predicting on test data.
"""pred_test00 = model000.predict(x_testcnn)
y_pred00 = encoder.inverse_transform(pred_test)
y_test00 = encoder.inverse_transform(y_test)

# Check for random predictions
df0 = pd.DataFrame(columns=['Predicted Labels', 'Actual Labels'])
df0['Predicted Labels'] = y_pred00.flatten()
df0['Actual Labels'] = y_test00.flatten()

df0.head(10)"""
```

Evalutation

```
[ ]: from sklearn.metrics import confusion_matrix,classification_report
     cm = confusion_matrix(y_test0, y_pred0)
     plt.figure(figsize = (12, 10))
     cm = pd.DataFrame(cm , index = [i for i in encoder.categories_] , columns = [i␣
      ↪for i in encoder.categories_])
     #cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
     sns.heatmap(cm, linecolor='white', cmap='Blues', linewidth=1, annot=True, fmt='.
      ↪2f')
     plt.title('Confusion Matrix', size=20)
     plt.xlabel('Predicted Labels', size=14)
     plt.ylabel('Actual Labels', size=14)
     plt.show()
     print(classification_report(y_test0, y_pred0))
```

Saving Best Model

```
[ ]: # MLP for Pima Indians Dataset Serialize to JSON and HDF5
     from tensorflow.keras.models import Sequential, model_from_json
     model_json = model.to_json()
     with open("CNN_model.json", "w") as json_file:
         json_file.write(model_json)

     model.save_weights("CNN_model_weights.h5")
     print("Saved model to disk")
```

```
[ ]: from tensorflow.keras.models import Sequential, model_from_json
     json_file = open('/kaggle/working/CNN_model.json', 'r')
     loaded_model_json = json_file.read()
     json_file.close()
     loaded_model = model_from_json(loaded_model_json)

     loaded_model.load_weights("/kaggle/working/best_model1_weights.h5")
     print("Loaded model from disk")
```

```
[ ]: loaded_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
      ↪metrics=['accuracy'])
     score = loaded_model.evaluate(x_testcnn,y_test)
     print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

pickle file

```
[ ]: import pickle

     with open('scaler2.pickle', 'wb') as f:
         pickle.dump(scaler, f)

     with open('scaler2.pickle', 'rb') as f:
```

```
    scaler2 = pickle.load(f)

with open('encoder2.pickle', 'wb') as f:
    pickle.dump(encoder, f)

with open('encoder2.pickle', 'rb') as f:
    encoder2 = pickle.load(f)

print("Done")
```

Test script

[ ]:
```
from tensorflow.keras.models import Sequential, model_from_json
json_file = open('/kaggle/working/CNN_model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

loaded_model.load_weights("/kaggle/working/best_model1_weights.h5")
print("Loaded model from disk")
```

[ ]:
```
import pickle

with open('/kaggle/working/scaler2.pickle', 'rb') as f:
    scaler2 = pickle.load(f)

with open('/kaggle/working/encoder2.pickle', 'rb') as f:
    encoder2 = pickle.load(f)

print("Done")
```

[ ]:
```
def zcr(data,frame_length,hop_length):
    zcr=librosa.feature.
 ↪zero_crossing_rate(data,frame_length=frame_length,hop_length=hop_length)
    return np.squeeze(zcr)
def rmse(data,frame_length=2048,hop_length=512):
    rmse=librosa.feature.
 ↪rms(data,frame_length=frame_length,hop_length=hop_length)
    return np.squeeze(rmse)
def mfcc(data,sr,frame_length=2048,hop_length=512,flatten:bool=True):
    mfcc=librosa.feature.mfcc(data,sr=sr)
    return np.squeeze(mfcc.T)if not flatten else np.ravel(mfcc.T)

def extract_features(data,sr=22050,frame_length=2048,hop_length=512):
    result=np.array([])

    result=np.hstack((result,
```

```
                        zcr(data,frame_length,hop_length),
                        rmse(data,frame_length,hop_length),
                        mfcc(data,sr,frame_length,hop_length)
                       ))
    return result
```

[ ]:
```
def get_predict_feat(path):
    d, s_rate= librosa.load(path, duration=2.5, offset=0.6)
    res=extract_features(d)
    result=np.array(res)
    result=np.reshape(result,newshape=(1,2376))
    i_result = scaler2.transform(result)
    final_result=np.expand_dims(i_result, axis=2)

    return final_result
```

[ ]:
```
res=get_predict_feat("/kaggle/input/ravdess-emotional-speech-audio/Actor_01/
 ↪03-01-07-01-01-01-01.wav")
print(res.shape)
```

[ ]:
```
emotions1={1:'Neutral', 2:'Calm', 3:'Happy', 4:'Sad', 5:'Angry', 6:'Fear', 7:
 ↪'Disgust',8:'Surprise'}
def prediction(path1):
    res=get_predict_feat(path1)
    predictions=loaded_model.predict(res)
    y_pred = encoder2.inverse_transform(predictions)
    print(y_pred[0][0])
```