# Let us import the Libraries required.

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

import urllib

# To use the model saved in the Json format, We are importing "model\_from\_json"

from tensorflow.keras.models import model\_from\_json

def mood(result):

if result=="Happy":

return 'Since you are happy, lets keep up the good mood with some amazing music!'

elif result=="Sad":

return 'It seems that you are having a bad day, lets cheer you up with some amazing music!'

elif result=="Disgust":

return 'It seems something has got you feeling disgusted. Lets improve your mood with some great music!'

elif result=="Neutral":

return 'It seems like a normal day. Lets turn it into a great one with some amazing music!'

elif result=="Fear":

return 'You seem very scared. We are sure that some music will help!'

elif result=="Angry":

return 'You seem angry. Listening to some music will surely help you calm down!'

elif result=="Surprise":

return 'You seem surprised! Hopefully its some good news. Lets celebrate it with some great music!'

def Emotion\_Analysis(img):

""" It does prediction of Emotions found in the Image provided, does the

Graphical visualisation, saves as Images and returns them """

# Read the Image through OpenCv's imread()

path = 'C:/EmotionChecker/Emotion-Investigator-master/Emotion-Investigator-master/static/images/' + str(img)

image = cv2.imread(path)

# Convert the Image into Gray Scale

gray\_frame = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Image size is reduced by 30% at each image scale.

scaleFactor = 1.3

# 5 neighbors should be present for each rectangle to be retained.

minNeighbors = 5

# Detect the Faces in the given Image and store it in faces.

faces = facec.detectMultiScale(gray\_frame, scaleFactor, minNeighbors)

# When Classifier could not detect any Face.

if len(faces) == 0:

return [img]

for (x, y, w, h) in faces:

# Taking the Face part in the Image as Region of Interest.

roi = gray\_frame[y:y+h, x:x+w]

# Let us resize the Image accordingly to use pretrained model.

roi = cv2.resize(roi, (48, 48))

# Let us make the Prediction of Emotion present in the Image

prediction = test\_model.predict\_emotion(

roi[np.newaxis, :, :, np.newaxis])

# Custom Symbols to print with text of emotion.

Symbols = {"Happy": ":)", "Sad": ":}", "Surprise": "!!",

"Angry": "?", "Disgust": "#", "Neutral": ".", "Fear": "~"}

## based on the prediction recommend music

# Defining the Parameters for putting Text on Image

Text = str(prediction) + Symbols[str(prediction)]

Text\_Color = (180, 105, 255)

Thickness = 2

Font\_Scale = 1

Font\_Type = cv2.FONT\_HERSHEY\_SIMPLEX

# Inserting the Text on Image

cv2.putText(image, Text, (x, y), Font\_Type,

Font\_Scale, Text\_Color, Thickness)

# Finding the Coordinates and Radius of Circle

xc = int((x + x+w)/2)

yc = int((y + y+h)/2)

radius = int(w/2)

# Drawing the Circle on the Image

cv2.circle(image, (xc, yc), radius, (0, 255, 0), Thickness)

# Saving the Predicted Image

predictedImagePath = "C:/EmotionChecker/Emotion-Investigator-master/Emotion-Investigator-master/static/images/pred\_" + str(img)

cv2.imwrite(predictedImagePath, image)

plt.gray()

plt.imshow(image)

plt.show()

# List of Emotions

EMOTIONS = ["Angry", "Disgust",

"Fear", "Happy",

"Neutral", "Sad",

"Surprise"]

# Finding the Probability of each Emotion

preds = test\_model.return\_probabs(roi[np.newaxis, :, :, np.newaxis])

# Converting the array into list

data = preds.tolist()[0]

# Initializing the Figure for Bar Graph

fig = plt.figure(figsize=(8, 5))

# Creating the bar plot

plt.bar(EMOTIONS, data, color='green',

width=0.4)

# Labelling the axes and title

plt.xlabel("Types of Emotions")

plt.ylabel("Probability")

plt.title("Facial Emotion Recognition")

emotionsList = []

for i in range(0,len(data)):

emotionsList.append(round(100\*data[i],2))

print(str(list(zip(EMOTIONS, emotionsList))))

# Saving the Bar Plot

path = "C:/EmotionChecker/Emotion-Investigator-master/Emotion-Investigator-master/static/images/" + "bar\_plot" + str(img)

plt.savefig(path)

# Returns a list containing the names of Original, Predicted, Bar Plot Images

return ([img, "pred" + img, "bar\_plot" + img, prediction])

class FacialExpressionModel(object):

""" A Class for Predicting the emotions using the pre-trained Model weights"""

EMOTIONS\_LIST = ["Angry", "Disgust",

"Fear", "Happy",

"Neutral", "Sad",

"Surprise"]

# Whenever we create an instance of class , these are initialized

def \_\_init\_\_(self, model\_json\_file, model\_weights\_file):

# Now Let us load model from JSON file which we created during Training

with open(model\_json\_file, "r") as json\_file:

# Reading the json file and storing it in loaded\_model

loaded\_model\_json = json\_file.read()

self.loaded\_model = model\_from\_json(loaded\_model\_json)

# Now, Let us load weights into the model

self.loaded\_model.load\_weights(model\_weights\_file)

def predict\_emotion(self, img):

""" It predicts the Emotion using our pre-trained model and returns it """

self.preds = self.loaded\_model.predict(img)

return FacialExpressionModel.EMOTIONS\_LIST[np.argmax(self.preds)]

def return\_probabs(self, img):

""" It returns the Probabilities of each emotions using pre-trained model """

self.preds = self.loaded\_model.predict(img)

return self.preds

# Creating an instance of the class with the parameters as model and its weights.

test\_model = FacialExpressionModel("C:/EmotionChecker2/model.json", "C:/EmotionChecker2/model\_weights.h5")

# Loading the classifier from the file.

facec = cv2.CascadeClassifier('C:/EmotionChecker2/haarcascade\_frontalface\_default.xml')

#####Main

captureJpeg = 's5.JPG'

result = Emotion\_Analysis(captureJpeg)

print(str(result))

sentence = mood(result[3])

print(str(sentence))