**app.py**

from flask import Flask, render\_template, Response, jsonify

import gunicorn

from camera import \*

app = Flask(\_\_name\_\_)

headings = ("Name","Album","Artist")

df1 = music\_rec()

df1 = df1.head(15)

@app.route('/')

def index():

print(df1.to\_json(orient='records'))

return render\_template('index.html', headings=headings, data=df1)

def gen(camera):

while True:

global df1

frame, df1 = camera.get\_frame()

yield (b'--frame\r\n'

b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video\_feed')

def video\_feed():

return Response(gen(VideoCamera()),

mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/t')

def gen\_table():

return df1.to\_json(orient='records')

if \_\_name\_\_ == '\_\_main\_\_':

app.debug = True

app.run()

**camera.py**

import numpy as np

import cv2

from PIL import Image

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from pandastable import Table, TableModel

from tensorflow.keras.preprocessing import image

import datetime

from threading import Thread

# from Spotipy import \*

import time

import pandas as pd

face\_cascade=cv2.CascadeClassifier("haarcascade\_frontalface\_default.xml")

ds\_factor=0.6

emotion\_model = Sequential()

emotion\_model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(48,48,1)))

emotion\_model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

emotion\_model.add(MaxPooling2D(pool\_size=(2, 2)))

emotion\_model.add(Dropout(0.25))

emotion\_model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

emotion\_model.add(MaxPooling2D(pool\_size=(2, 2)))

emotion\_model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

emotion\_model.add(MaxPooling2D(pool\_size=(2, 2)))

emotion\_model.add(Dropout(0.25))

emotion\_model.add(Flatten())

emotion\_model.add(Dense(1024, activation='relu'))

emotion\_model.add(Dropout(0.5))

emotion\_model.add(Dense(7, activation='softmax'))

emotion\_model.load\_weights('D:\ADP\ADP-2\Emotion-Music-Recommendation-1\Emotion-Music-Recommendation\model.h5')

cv2.ocl.setUseOpenCL(False)

emotion\_dict = {0:"Angry",1:"Disgusted",2:"Fearful",3:"Happy",4:"Neutral",5:"Sad",6:"Surprised"}

music\_dist={0:"songs/angry.csv",1:"songs/disgusted.csv ",2:"songs/fearful.csv",3:"songs/happy.csv",4:"songs/neutral.csv",5:"songs/sad.csv",6:"songs/surprised.csv"}

global last\_frame1

last\_frame1 = np.zeros((480, 640, 3), dtype=np.uint8)

global cap1

show\_text=[0]

''' Class for calculating FPS while streaming. Used this to check performance of using another thread for video streaming '''

class FPS:

def \_\_init\_\_(self):

# store the start time, end time, and total number of frames

# that were examined between the start and end intervals

self.\_start = None

self.\_end = None

self.\_numFrames = 0

def start(self):

# start the timer

self.\_start = datetime.datetime.now()

return self

def stop(self):

# stop the timer

self.\_end = datetime.datetime.now()

def update(self):

# increment the total number of frames examined during the

# start and end intervals

self.\_numFrames += 1

def elapsed(self):

# return the total number of seconds between the start and

# end interval

return (self.\_end - self.\_start).total\_seconds()

def fps(self):

# compute the (approximate) frames per second

return self.\_numFrames / self.elapsed()

''' Class for using another thread for video streaming to boost performance '''

class WebcamVideoStream:

def \_\_init\_\_(self, src=0):

self.stream = cv2.VideoCapture(src,cv2.CAP\_DSHOW)

(self.grabbed, self.frame) = self.stream.read()

self.stopped = False

def start(self):

# start the thread to read frames from the video stream

Thread(target=self.update, args=()).start()

return self

def update(self):

# keep looping infinitely until the thread is stopped

while True:

# if the thread indicator variable is set, stop the thread

if self.stopped:

return

# otherwise, read the next frame from the stream

(self.grabbed, self.frame) = self.stream.read()

def read(self):

# return the frame most recently read

return self.frame

def stop(self):

# indicate that the thread should be stopped

self.stopped = True

''' Class for reading video stream, generating prediction and recommendations '''

class VideoCamera(object):

def get\_frame(self):

global cap1

global df1

cap1 = WebcamVideoStream(src=0).start()

image = cap1.read()

image=cv2.resize(image,(600,500))

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

face\_rects=face\_cascade.detectMultiScale(gray,1.3,5)

df1 = pd.read\_csv(music\_dist[show\_text[0]])

df1 = df1[['Name','Album','Artist']]

df1 = df1.head(15)

for (x,y,w,h) in face\_rects:

cv2.rectangle(image,(x,y-50),(x+w,y+h+10),(0,255,0),2)

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

cropped\_img = np.expand\_dims(np.expand\_dims(cv2.resize(roi\_gray\_frame, (48, 48)), -1), 0)

prediction = emotion\_model.predict(cropped\_img)

maxindex = int(np.argmax(prediction))

show\_text[0] = maxindex

#print("===========================================",music\_dist[show\_text[0]],"===========================================")

#print(df1)

cv2.putText(image, emotion\_dict[maxindex], (x+20, y-60), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)

df1 = music\_rec()

global last\_frame1

last\_frame1 = image.copy()

pic = cv2.cvtColor(last\_frame1, cv2.COLOR\_BGR2RGB)

img = Image.fromarray(last\_frame1)

img = np.array(img)

ret, jpeg = cv2.imencode('.jpg', img)

return jpeg.tobytes(), df1

def music\_rec():

# print('---------------- Value ------------', music\_dist[show\_text[0]])

df = pd.read\_csv(music\_dist[show\_text[0]])

df = df[['Name','Album','Artist']]

df = df.head(15)

return df

**Spotipy.py**

import spotipy

import spotipy.oauth2 as oauth2

from spotipy.oauth2 import SpotifyOAuth

from spotipy.oauth2 import SpotifyClientCredentials

import pandas as pd

import time

auth\_manager = SpotifyClientCredentials('','')

sp = spotipy.Spotify(auth\_manager=auth\_manager)

def getTrackIDs(user, playlist\_id):

track\_ids = []

playlist = sp.user\_playlist(user, playlist\_id)

for item in playlist['tracks']['items']:

track = item['track']

track\_ids.append(track['id'])

return track\_ids

def getTrackFeatures(id):

track\_info = sp.track(id)

name = track\_info['name']

album = track\_info['album']['name']

artist = track\_info['album']['artists'][0]['name']

# release\_date = track\_info['album']['release\_date']

# length = track\_info['duration\_ms']

# popularity = track\_info['popularity']

track\_data = [name, album, artist] #, release\_date, length, popularity

return track\_data

# Code for creating dataframe of feteched playlist

emotion\_dict = {0:"Angry",1:"Disgusted",2:"Fearful",3:"Happy",4:"Neutral",5:"Sad",6:"Surprised"}

music\_dist={0:"0l9dAmBrUJLylii66JOsHB?si=e1d97b8404e34343",1:"1n6cpWo9ant4WguEo91KZh?si=617ea1c66ab6446b ",2:"4cllEPvFdoX6NIVWPKai9I?si=dfa422af2e8448ef",3:"0deORnapZgrxFY4nsKr9JA?si=7a5aba992ea14c93",4:"4kvSlabrnfRCQWfN0MgtgA?si=b36add73b4a74b3a",5:"1n6cpWo9ant4WguEo91KZh?si=617ea1c66ab6446b",6:"37i9dQZEVXbMDoHDwVN2tF?si=c09391805b6c4651"}

'''

Code can def be modularised into a function but i tried to write it when i was extremely sleepy so thought screw it and repeated code block

Uncomment for fetching updated playlists

'''

# track\_ids = getTrackIDs('spotify',music\_dist[0])

# track\_list = []

# for i in range(len(track\_ids)):

# time.sleep(.3)

# track\_data = getTrackFeatures(track\_ids[i])

# track\_list.append(track\_data)

# df = pd.DataFrame(track\_list, columns = ['Name','Album','Artist']) # ,'Release\_date','Length','Popularity'

# df.to\_csv('songs/angry.csv')

# print("CSV Generated")

# track\_ids = getTrackIDs('spotify',music\_dist[1])

# track\_list = []

# for i in range(len(track\_ids)):

# time.sleep(.3)

# track\_data = getTrackFeatures(track\_ids[i])

# track\_list.append(track\_data)

# df = pd.DataFrame(track\_list, columns = ['Name','Album','Artist']) # ,'Release\_date','Length','Popularity'

# df.to\_csv('songs/disgusted.csv')

# print("CSV Generated")

# track\_ids = getTrackIDs('spotify',music\_dist[2])

# track\_list = []

# for i in range(len(track\_ids)):

# time.sleep(.3)

# track\_data = getTrackFeatures(track\_ids[i])

# track\_list.append(track\_data)

# df = pd.DataFrame(track\_list, columns = ['Name','Album','Artist']) # ,'Release\_date','Length','Popularity'

# df.to\_csv('songs/fearful.csv')

# print("CSV Generated")

# track\_ids = getTrackIDs('spotify',music\_dist[3])

# track\_list = []

# for i in range(len(track\_ids)):

# time.sleep(.3)

# track\_data = getTrackFeatures(track\_ids[i])

# track\_list.append(track\_data)

# df = pd.DataFrame(track\_list, columns = ['Name','Album','Artist']) # ,'Release\_date','Length','Popularity'

# df.to\_csv('songs/happy.csv')

# print("CSV Generated")

# track\_ids = getTrackIDs('spotify',music\_dist[4])

# track\_list = []

# for i in range(len(track\_ids)):

# time.sleep(.3)

# track\_data = getTrackFeatures(track\_ids[i])

# track\_list.append(track\_data)

# df = pd.DataFrame(track\_list, columns = ['Name','Album','Artist']) # ,'Release\_date','Length','Popularity'

# df.to\_csv('songs/neutral.csv')

# print("CSV Generated")

# track\_ids = getTrackIDs('spotify',music\_dist[5])

# track\_list = []

# for i in range(len(track\_ids)):

# time.sleep(.3)

# track\_data = getTrackFeatures(track\_ids[i])

# track\_list.append(track\_data)

# df = pd.DataFrame(track\_list, columns = ['Name','Album','Artist']) # ,'Release\_date','Length','Popularity'

# df.to\_csv('songs/sad.csv')

# print("CSV Generated")

# track\_ids = getTrackIDs('spotify',music\_dist[6])

# track\_list = []

# for i in range(len(track\_ids)):

# time.sleep(.3)

# track\_data = getTrackFeatures(track\_ids[i])

# track\_list.append(track\_data)

# df = pd.DataFrame(track\_list, columns = ['Name','Album','Artist']) # ,'Release\_date','Length','Popularity'

# df.to\_csv('songs/surprised.csv')

# print("CSV Generated")