import datetime as dt

import matplotlib.pyplot as plt

import matplotlib.animation as animation

import requests

import numpy as np

from EAR\_calculator import \*

from imutils import face\_utils

from imutils.video import VideoStream

import imutils

import dlib

import time

import argparse

import cv2

import pandas as pd

import csv

from playsound import playsound

from scipy.spatial import distance as dist

import os

from datetime import datetime

# Creating the dataset

def assure\_path\_exists(path):

dir = os.path.dirname(path)

if not os.path.exists(dir):

os.makedirs(dir)

#all eye and mouth aspect ratio with time

ear\_list=[]

total\_ear=[]

mar\_list=[]

total\_mar=[]

ts=[]

total\_ts=[]

url = "http://<YOUR\_IP\_HERE>/shot.jpg"

# Construct the argument parser and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-p", "--shape\_predictor", required = True, help = "path to dlib's facial landmark predictor")

ap.add\_argument("-r", "--picamera", type = int, default = -1, help = "whether raspberry pi camera shall be used or not")

args = vars(ap.parse\_args())

# Declare a constant which will work as the threshold for EAR value, below which it will be regared as a blink

EAR\_THRESHOLD = 0.3

# Declare another costant to hold the consecutive number of frames to consider for a blink

CONSECUTIVE\_FRAMES = 15

# Another constant which will work as a threshold for MAR value

MAR\_THRESHOLD = 14

# Initialize two counters

BLINK\_COUNT = 0

FRAME\_COUNT = 0

#Now, intialize the dlib's face detector model as 'detector' and the landmark predictor model as 'predictor'

print("[INFO]Loading the predictor.....")

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor(args["shape\_predictor"])

# Grab the indexes of the facial landamarks for the left and right eye respectively

(lstart, lend) = face\_utils.FACIAL\_LANDMARKS\_IDXS["left\_eye"]

(rstart, rend) = face\_utils.FACIAL\_LANDMARKS\_IDXS["right\_eye"]

(mstart, mend) = face\_utils.FACIAL\_LANDMARKS\_IDXS["mouth"]

# Now start the video stream and allow the camera to warm-up

print("[INFO]Loading Camera.....")

time.sleep(2)

assure\_path\_exists("dataset\_phonecam/")

count\_sleep = 0

count\_yawn = 0

fig = plt.figure()

ax = fig.add\_subplot(1, 1, 1)

xs = []

ys = []

while True:

img\_resp = requests.get(url)

img\_arr = np.array(bytearray(img\_resp.content), dtype = np.uint8)

frame = cv2.imdecode(img\_arr, -1)

frame = imutils.resize(frame, width = 875)

frame = cv2.rotate(frame, cv2.ROTATE\_90\_CLOCKWISE)

cv2.putText(frame, "PRESS 'q' TO EXIT", (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 3)

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

# Detect faces

rects = detector(frame, 1)

# Now loop over all the face detections and apply the predictor

for (i, rect) in enumerate(rects):

shape = predictor(gray, rect)

# Convert it to a (68, 2) size numpy array

shape = face\_utils.shape\_to\_np(shape)

# Draw a rectangle over the detected face

(x, y, w, h) = face\_utils.rect\_to\_bb(rect)

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Put a number

cv2.putText(frame, "Driver", (x - 10, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

leftEye = shape[lstart:lend]

rightEye = shape[rstart:rend]

mouth = shape[mstart:mend]

# Compute the EAR for both the eyes

leftEAR = eye\_aspect\_ratio(leftEye)

rightEAR = eye\_aspect\_ratio(rightEye)

# Take the average of both the EAR

EAR = (leftEAR + rightEAR) / 2.0

#live datawrite in csv

ear\_list.append(EAR)

ts.append(dt.datetime.now().strftime('%H:%M:%S.%f'))

# Compute the convex hull for both the eyes and then visualize it

leftEyeHull = cv2.convexHull(leftEye)

rightEyeHull = cv2.convexHull(rightEye)

# Draw the contours

cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)

cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

cv2.drawContours(frame, [mouth], -1, (0, 255, 0), 1)

MAR = mouth\_aspect\_ratio(mouth)

mar\_list.append(MAR/10)

# Check if EAR < EAR\_THRESHOLD, if so then it indicates that a blink is taking place

# Thus, count the number of frames for which the eye remains closed

if EAR < EAR\_THRESHOLD:

FRAME\_COUNT += 1

cv2.drawContours(frame, [leftEyeHull], -1, (0, 0, 255), 1)

cv2.drawContours(frame, [rightEyeHull], -1, (0, 0, 255), 1)

if FRAME\_COUNT >= CONSECUTIVE\_FRAMES:

count\_sleep += 1

# Add the frame to the dataset ar a proof of drowsy driving

cv2.imwrite("dataset\_phonecam/frame\_sleep%d.jpg" % count\_sleep, frame)

playsound('sound files/alarm.mp3')

cv2.putText(frame, "DROWSINESS ALERT!", (270, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

else:

if FRAME\_COUNT >= CONSECUTIVE\_FRAMES:

playsound('sound files/warning.mp3')

FRAME\_COUNT = 0

# Check if the person is yawning

if MAR > MAR\_THRESHOLD:

count\_yawn += 1

cv2.drawContours(frame, [mouth], -1, (0, 0, 255), 1)

cv2.putText(frame, "DROWSINESS ALERT!", (270, 300), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

cv2.imwrite("dataset\_phonecam/frame\_yawn%d.jpg" % count\_yawn, frame)

playsound('sound files/alarm.mp3')

playsound('sound files/warning\_yawn.mp3')

#total data collection for plotting

for i in ear\_list:

total\_ear.append(i)

for i in mar\_list:

total\_mar.append(i)

for i in ts:

total\_ts.append(i)

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

if key == ord('q'):

break

a = total\_ear

b=total\_mar

c = total\_ts

df = pd.DataFrame({"EAR" : a, "MAR":b,"TIME" : c})

df.to\_csv("op\_phonecam.csv", index=False)

df=pd.read\_csv("op\_phonecam.csv")

df.plot(x='TIME',y=['EAR','MAR'])

#plt.xticks(rotation=45, ha='right')

plt.subplots\_adjust(bottom=0.30)

plt.title('EAR & MAR calculation over time of phone cam')

plt.ylabel('EAR & MAR')

plt.gca().axes.get\_xaxis().set\_visible(False)

plt.show()

cv2.destroyAllWindows()