# USAGE

# python detect\_blinks.py --shape-predictor shape\_predictor\_68\_face\_landmarks.dat --video blink\_detection\_demo.mp4

# python detect\_blinks.py --shape-predictor shape\_predictor\_68\_face\_landmarks.dat

# import the necessary packages

from scipy.spatial import distance as dist

from imutils.video import FileVideoStream

from imutils.video import VideoStream

from imutils import face\_utils

import numpy as np

import argparse

import imutils

import time

import dlib

import cv2

def eye\_aspect\_ratio(eye):

# compute the euclidean distances between the two sets of

# vertical eye landmarks (x, y)-coordinates

A = dist.euclidean(eye[1], eye[5])

B = dist.euclidean(eye[2], eye[4])

# compute the euclidean distance between the horizontal

# eye landmark (x, y)-coordinates

C = dist.euclidean(eye[0], eye[3])

# compute the eye aspect ratio

ear = (A + B) / (2.0 \* C)

# return the eye aspect ratio

return ear

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-p", "--shape-predictor", required=True,

help="path to facial landmark predictor")

ap.add\_argument("-v", "--video", type=str, default="",

help="path to input video file")

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

# define two constants, one for the eye aspect ratio to indicate

# blink and then a second constant for the number of consecutive

# frames the eye must be below the threshold

EYE\_AR\_THRESH = 0.25

EYE\_AR\_SHORT = 2

EYE\_AR\_LONG = 7

# initialize the frame counters and the total number of blinks

COUNTER = 0

TOTAL\_SHORT = 0

TOTAL\_LONG = 0

# initialize dlib's face detector (HOG-based) and then create

# the facial landmark predictor

print("[INFO] loading facial landmark predictor...")

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

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

# grab the indexes of the facial landmarks 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"]

# start the video stream thread

print("[INFO] starting video stream thread...")

vs = FileVideoStream(args["video"]).start()

fileStream = True

vs = VideoStream(src=0).start()

# vs = VideoStream(usePiCamera=True).start()

fileStream = False

time.sleep(1.0)

# loop over frames from the video stream

while True:

# if this is a file video stream, then we need to check if

# there any more frames left in the buffer to process

if fileStream and not vs.more():

break

# grab the frame from the threaded video file stream, resize

# it, and convert it to grayscale

# channels)

frame = vs.read()

frame = imutils.resize(frame, width=450)

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

# detect faces in the grayscale frame

rects = detector(gray, 0)

# loop over the face detections

for rect in rects:

# determine the facial landmarks for the face region, then

# convert the facial landmark (x, y)-coordinates to a NumPy

# array

shape = predictor(gray, rect)

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

# extract the left and right eye coordinates, then use the

# coordinates to compute the eye aspect ratio for both eyes

leftEye = shape[lStart:lEnd]

rightEye = shape[rStart:rEnd]

leftEAR = eye\_aspect\_ratio(leftEye)

rightEAR = eye\_aspect\_ratio(rightEye)

# average the eye aspect ratio together for both eyes

ear = (leftEAR + rightEAR) / 2.0

# compute the convex hull for the left and right eye, then

# visualize each of the eyes

leftEyeHull = cv2.convexHull(leftEye)

rightEyeHull = cv2.convexHull(rightEye)

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

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

# check to see if the eye aspect ratio is below the blink

# threshold, and if so, increment the blink frame counter

if ear < EYE\_AR\_THRESH:

COUNTER += 1

# otherwise, the eye aspect ratio is not below the blink

# threshold

else:

# if the eyes were closed for a sufficient number of

# then increment the total number of blinks

if ((COUNTER >= EYE\_AR\_SHORT) and (COUNTER <=EYE\_AR\_LONG)):

TOTAL\_SHORT += 1

print ("\n0")

if(COUNTER > EYE\_AR\_LONG):

TOTAL\_LONG +=1

print ("\n1")

# reset the eye frame counter

COUNTER = 0

# draw the total number of blinks on the frame along with

# the computed eye aspect ratio for the frame

cv2.putText(frame, "Blink Count long: {}".format(TOTAL\_LONG), (30, 30),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (255, 255, 255), 2)

cv2.putText(frame, "Blink Count short: {}".format(TOTAL\_SHORT), (30, 60),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (255, 255, 255), 2)

cv2.putText(frame, "Ratio: {:.2f}".format(ear), (300, 30),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (40, 80, 255), 2)

# show the frame

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop

if key == ord("q"):

break

# do a bit of cleanup

cv2.destroyAllWindows()

vs.stop()