import os

import cv2

import face\_recognition

import numpy as np

from tqdm import tqdm

from collections import defaultdict

from imutils.video import VideoStream

from eye\_status import \*

def init():

face\_cascPath = 'haarcascade\_frontalface\_alt.xml'

# face\_cascPath = 'lbpcascade\_frontalface.xml'

open\_eye\_cascPath = 'haarcascade\_eye\_tree\_eyeglasses.xml'

left\_eye\_cascPath = 'haarcascade\_lefteye\_2splits.xml'

right\_eye\_cascPath ='haarcascade\_righteye\_2splits.xml'

dataset = 'faces'

face\_detector = cv2.CascadeClassifier(face\_cascPath)

open\_eyes\_detector = cv2.CascadeClassifier(open\_eye\_cascPath)

left\_eye\_detector = cv2.CascadeClassifier(left\_eye\_cascPath)

right\_eye\_detector = cv2.CascadeClassifier(right\_eye\_cascPath)

print("[LOG] Opening webcam ...")

video\_capture = VideoStream(src=0).start()

model = load\_model()

print("[LOG] Collecting images ...")

images = []

for direc, \_, files in tqdm(os.walk(dataset)):

for file in files:

if file.endswith("jpg"):

images.append(os.path.join(direc,file))

return (model,face\_detector, open\_eyes\_detector, left\_eye\_detector,right\_eye\_detector, video\_capture, images)

def process\_and\_encode(images):

# initialize the list of known encodings and known names

known\_encodings = []

known\_names = []

print("[LOG] Encoding faces ...")

for image\_path in tqdm(images):

# Load image

image = cv2.imread(image\_path)

# Convert it from BGR to RGB

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# detect face in the image and get its location (square boxes coordinates)

boxes = face\_recognition.face\_locations(image, model='hog')

# Encode the face into a 128-d embeddings vector

encoding = face\_recognition.face\_encodings(image, boxes)

# the person's name is the name of the folder where the image comes from

name = image\_path.split(os.path.sep)[-2]

if len(encoding) > 0 :

known\_encodings.append(encoding[0])

known\_names.append(name)

return {"encodings": known\_encodings, "names": known\_names}

def isBlinking(history, maxFrames):

""" @history: A string containing the history of eyes status

where a '1' means that the eyes were closed and '0' open.

@maxFrames: The maximal number of successive frames where an eye is closed """

for i in range(maxFrames):

pattern = '1' + '0'\*(i+1) + '1'

if pattern in history:

return True

return False

def detect\_and\_display(model, video\_capture, face\_detector, open\_eyes\_detector, left\_eye\_detector, right\_eye\_detector, data, eyes\_detected):

frame = video\_capture.read()

# resize the frame

frame = cv2.resize(frame, (0, 0), fx=0.6, fy=0.6)

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

rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Detect faces

faces = face\_detector.detectMultiScale(

gray,

scaleFactor=1.2,

minNeighbors=5,

minSize=(50, 50),

flags=cv2.CASCADE\_SCALE\_IMAGE

)

# for each detected face

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

# Encode the face into a 128-d embeddings vector

encoding = face\_recognition.face\_encodings(rgb, [(y, x+w, y+h, x)])[0]

# Compare the vector with all known faces encodings

matches = face\_recognition.compare\_faces(data["encodings"], encoding)

# For now we don't know the person name

name = "Unknown"

# If there is at least one match:

if True in matches:

matchedIdxs = [i for (i, b) in enumerate(matches) if b]

counts = {}

for i in matchedIdxs:

name = data["names"][i]

counts[name] = counts.get(name, 0) + 1

# determine the recognized face with the largest number of votes

name = max(counts, key=counts.get)

face = frame[y:y+h,x:x+w]

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

eyes = []

# Eyes detection

# check first if eyes are open (with glasses taking into account)

open\_eyes\_glasses = open\_eyes\_detector.detectMultiScale(

gray\_face,

scaleFactor=1.1,

minNeighbors=5,

minSize=(30, 30),

flags = cv2.CASCADE\_SCALE\_IMAGE

)

# if open\_eyes\_glasses detect eyes then they are open

if len(open\_eyes\_glasses) == 2:

eyes\_detected[name]+='1'

for (ex,ey,ew,eh) in open\_eyes\_glasses:

cv2.rectangle(face,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

# otherwise try detecting eyes using left and right\_eye\_detector

# which can detect open and closed eyes

else:

# separate the face into left and right sides

left\_face = frame[y:y+h, x+int(w/2):x+w]

left\_face\_gray = gray[y:y+h, x+int(w/2):x+w]

right\_face = frame[y:y+h, x:x+int(w/2)]

right\_face\_gray = gray[y:y+h, x:x+int(w/2)]

# Detect the left eye

left\_eye = left\_eye\_detector.detectMultiScale(

left\_face\_gray,

scaleFactor=1.1,

minNeighbors=5,

minSize=(30, 30),

flags = cv2.CASCADE\_SCALE\_IMAGE

)

# Detect the right eye

right\_eye = right\_eye\_detector.detectMultiScale(

right\_face\_gray,

scaleFactor=1.1,

minNeighbors=5,

minSize=(30, 30),

flags = cv2.CASCADE\_SCALE\_IMAGE

)

eye\_status = '1' # we suppose the eyes are open

# For each eye check wether the eye is closed.

# If one is closed we conclude the eyes are closed

for (ex,ey,ew,eh) in right\_eye:

color = (0,255,0)

pred = predict(right\_face[ey:ey+eh,ex:ex+ew],model)

if pred == 'closed':

eye\_status='0'

color = (0,0,255)

cv2.rectangle(right\_face,(ex,ey),(ex+ew,ey+eh),color,2)

for (ex,ey,ew,eh) in left\_eye:

color = (0,255,0)

pred = predict(left\_face[ey:ey+eh,ex:ex+ew],model)

if pred == 'closed':

eye\_status='0'

color = (0,0,255)

cv2.rectangle(left\_face,(ex,ey),(ex+ew,ey+eh),color,2)

eyes\_detected[name] += eye\_status

# Each time, we check if the person has blinked

# If yes, we display its name

if isBlinking(eyes\_detected[name],3):

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

# Display name

y = y - 15 if y - 15 > 15 else y + 15

cv2.putText(frame, name, (x, y), cv2.FONT\_HERSHEY\_SIMPLEX,0.75, (0, 255, 0), 2)

return frame

if \_\_name\_\_ == "\_\_main\_\_":

(model, face\_detector, open\_eyes\_detector,left\_eye\_detector,right\_eye\_detector, video\_capture, images) = init()

data = process\_and\_encode(images)

eyes\_detected = defaultdict(str)

while True:

frame = detect\_and\_display(model, video\_capture, face\_detector, open\_eyes\_detector,left\_eye\_detector,right\_eye\_detector, data, eyes\_detected)

cv2.imshow("Face Liveness Detector", frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows()

video\_capture.stop()