import encodings

from itertools import count

import face\_recognition

import cv2

import numpy as np

import datetime

import csv

# This is a demo of running face recognition on live video from your webcam. It's a little more complicated than the

# other example, but it includes some basic performance tweaks to make things run a lot faster:

# 1. Process each video frame at 1/4 resolution (though still display it at full resolution)

# 2. Only detect faces in every other frame of video.

# PLEASE NOTE: This example requires OpenCV (the `cv2` library) to be installed only to read from your webcam.

# OpenCV is \*not\* required to use the face\_recognition library. It's only required if you want to run this

# specific demo. If you have trouble installing it, try any of the other demos that don't require it instead.

# Get a reference to webcam #0 (the default one)

# import the necessary packages

from tensorflow.keras.applications.mobilenet\_v2 import preprocess\_input

from tensorflow.keras.preprocessing.image import img\_to\_array

from tensorflow.keras.models import load\_model

from imutils.video import VideoStream

import numpy as np

import imutils

import time

import cv2

import os

def detect\_and\_predict\_mask(frame, faceNet, maskNet):

# grab the dimensions of the frame and then construct a blob

# from it

(h, w) = frame.shape[:2]

blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),

(104.0, 177.0, 123.0))

# pass the blob through the network and obtain the face detections

faceNet.setInput(blob)

detections = faceNet.forward()

print(detections.shape)

# initialize our list of faces, their corresponding locations,

# and the list of predictions from our face mask network

faces = []

locs = []

preds = []

# loop over the detections

for i in range(0, detections.shape[2]):

# extract the confidence (i.e., probability) associated with

# the detection

confidence = detections[0, 0, i, 2]

# filter out weak detections by ensuring the confidence is

# greater than the minimum confidence

if confidence > 0.5:

# compute the (x, y)-coordinates of the bounding box for

# the object

box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

(startX, startY, endX, endY) = box.astype("int")

# ensure the bounding boxes fall within the dimensions of

# the frame

(startX, startY) = (max(0, startX), max(0, startY))

(endX, endY) = (min(w - 1, endX), min(h - 1, endY))

# extract the face ROI, convert it from BGR to RGB channel

# ordering, resize it to 224x224, and preprocess it

face = frame[startY:endY, startX:endX]

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

face = cv2.resize(face, (224, 224))

face = img\_to\_array(face)

face = preprocess\_input(face)

# add the face and bounding boxes to their respective

# lists

faces.append(face)

locs.append((startX, startY, endX, endY))

# only make a predictions if at least one face was detected

if len(faces) > 0:

# for faster inference we'll make batch predictions on \*all\*

# faces at the same time rather than one-by-one predictions

# in the above `for` loop

faces = np.array(faces, dtype="float32")

preds.append(maskNet.predict(faces, batch\_size=32)[0].tolist())

# return a 2-tuple of the face locations and their corresponding

# locations

return (locs, preds)

video\_capture = cv2.VideoCapture(0)

img\_path = os.getcwd() + "//Recog\_Train"

images = []

known\_face\_names = []

known\_face\_encodings = []

encode\_list\_cl = []

myList = os.listdir(img\_path)

#print(myList)

for subdir in os.listdir(img\_path):

path = img\_path + '/' + subdir

for subdir in os.listdir(path):

path1 = path + '/' + subdir

path1 = path1 + '/'

for img in os.listdir(path1):

img\_pic = path1 + img

known\_face\_names.append(subdir)

cur\_img = cv2.imread(img\_pic)

images.append(cur\_img)

def find\_encodings(images) :

#for names in images :

for img in images :

encodings = face\_recognition.face\_encodings(img)[0]

known\_face\_encodings.append(encodings)

return known\_face\_encodings

encodeListKnown = find\_encodings(images)

# # Load a sample picture and learn how to recognize it.

# Saad\_image = face\_recognition.load\_image\_file("Saad.jpeg")

# Saad\_face\_encoding = face\_recognition.face\_encodings(Saad\_image)[0]

# # Load a sample picture and learn how to recognize it.

# # Abdullah\_image = face\_recognition.load\_image\_file("Abdullah.jpeg")

# # Abdullah\_face\_encoding = face\_recognition.face\_encodings(Abdullah\_image)[0]

# # Load a second sample picture and learn how to recognize it.

# Ayesha\_image = face\_recognition.load\_image\_file("Ayesha.jpeg")

# Ayesha\_face\_encoding = face\_recognition.face\_encodings(Ayesha\_image)[0]

# Create arrays of known face encodings and their names

# known\_face\_encodings = [

# Saad\_face\_encoding,

# # Abdullah\_face\_encoding,

# Ayesha\_face\_encoding

# ]

# known\_face\_names = [

# "Saad",

# # "Abdullah",

# "Ayesha"

count=0

tcount=0

# Initialize some variables

face\_locations = []

face\_encodings = []

face\_names = []

process\_this\_frame = True

# load our serialized face detector model from disk

prototxtPath = r"face\_detector\deploy.prototxt"

weightsPath = r"face\_detector\res10\_300x300\_ssd\_iter\_140000.caffemodel"

faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk

maskNet = load\_model("mask\_detector.model")

# initialize the video stream

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

name1=""

while True:

# Grab a single frame of video

ret, frame = video\_capture.read()

# Resize frame of video to 1/4 size for faster face recognition processing

small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

# Convert the image from BGR color (which OpenCV uses) to RGB color (which face\_recognition uses)

rgb\_small\_frame = small\_frame[:, :, ::-1]

# Only process every other frame of video to save time

if process\_this\_frame:

# Find all the faces and face encodings in the current frame of video

face\_locations = face\_recognition.face\_locations(rgb\_small\_frame)

face\_encodings = face\_recognition.face\_encodings(

rgb\_small\_frame, face\_locations)

face\_names = []

for face\_encoding in face\_encodings:

# See if the face is a match for the known face(s)

matches = face\_recognition.compare\_faces(

known\_face\_encodings, face\_encoding)

# # If a match was found in known\_face\_encodings, just use the first one.

# if True in matches:

# first\_match\_index = matches.index(True)

# name = known\_face\_names[first\_match\_index]

# Or instead, use the known face with the smallest distance to the new face

name = "Unknown"

face\_distances = face\_recognition.face\_distance(

known\_face\_encodings, face\_encoding)

best\_match\_index = np.argmin(face\_distances)

tcount=+1

if matches[best\_match\_index]:

name = known\_face\_names[best\_match\_index]

if(name==name1):

count=+1

if(name!=name1):

with open('.\Attendance.csv', 'a') as f:

date\_time\_string = datetime.datetime.now().strftime("%y/%m/%d %H:%M:%S")

f.writelines(f'\n{name},{date\_time\_string}')

name1=name

face\_names.append(name)

print(name)

with open('.\Accuracy.csv', 'a') as f:

f.writelines(f'/n{name}')

# csv\_reader = csv.reader(f, delimiter=',')

# count = 0

# for row in csv\_reader:

# f.writelines(f'\n{name},')

# for row in f:

# count=+1

# print("total: ",count)

process\_this\_frame = not process\_this\_frame

# Display the results

for (top, right, bottom, left), name in zip(face\_locations, face\_names):

# Scale back up face locations since the frame we detected in was scaled to 1/4 size

top \*= 4

right \*= 4

bottom \*= 4

left \*= 4

# Draw a box around the face

if name == "Unknown":

color = (0, 0, 255)

else:

color = (0, 255, 0)

cv2.rectangle(frame, (left, top), (right, bottom), color, 2)

(locs, preds) = detect\_and\_predict\_mask(frame, faceNet, maskNet)

for (box, pred) in zip(locs, preds):

(startX, startY, endX, endY) = box

(mask, withoutMask) = pred

label = "Mask" if mask > withoutMask else "No Mask"

color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

label = "{}: {:.2f}%".format(label, max(mask, withoutMask) \* 100)

cv2.putText(frame, label, (startX, startY - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, color, 2)

cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

print((count/tcount)\*100)

# Draw a label with a name below the face

cv2.rectangle(frame, (left, bottom - 35),

(right, bottom), color, cv2.FILLED)

font = cv2.FONT\_HERSHEY\_DUPLEX

cv2.putText(frame, name,(left + 6, bottom - 6),

font, 1.0, (255, 255, 255), 1)

# show the output frame

# Display the resulting image

cv2.imshow('Video', frame)

# Hit 'q' on the keyboard to quit!

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

break

# Release handle to the webcam

video\_capture.release()

cv2.destroyAllWindows()