import numpy as np

import cv2

import time

import os

label\_text = []

person\_names = [""]

def convertToRGB(img):

return cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

def convertToGray(img):

return cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

#draw rectangle on image

def draw\_rectangle(img, rect):

#print(len(img))

#for rect in rects:

(x, y, w, h) = rect

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

#drawing text from x and y cordinate

def draw\_text(img, text, x, y):

cv2.putText(img, text, (x, y), cv2.FONT\_HERSHEY\_PLAIN, 3.5, (200, 255, 0), 5)

#face detecting

def face\_detect(img, scaleFactor=1.3, minNeighbors=5, returnGray=True):

img\_copy = img.copy()

gray = convertToGray(img\_copy)

#face\_cascade = cv2.CascadeClassifier('opencv-files/haarcascade\_frontalface\_alt.xml')

face\_cascade = cv2.CascadeClassifier('opencv-files/lbpcascade\_frontalface.xml')

faces = face\_cascade.detectMultiScale(gray, scaleFactor, minNeighbors);

if (len(faces) == 0):

return None, None

#print(faces[0])

#print(gray)

result = []

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

result.append((x, y, w, h))

if returnGray:

return gray, result

else:

return img, result

#(x, y, w, h) = faces[0]

#return gray[y:y+w, x:x+h], faces[0]

#reading training data

def reading\_training\_data(training\_folder\_path):

dirs = os.listdir(training\_folder\_path)

faces = []

labels = []

for dir\_name in dirs:

person\_names.append(dir\_name.split("-")[1])

subject\_dir\_path = training\_folder\_path +"/"+dir\_name

images\_names = os.listdir(subject\_dir\_path)

for image\_name in images\_names:

if image\_name.startswith("."):

continue

#image for each images

image\_path = subject\_dir\_path+"/"+image\_name

label = int(dir\_name.split("-")[0].replace("person",""))

label\_text.append(person\_names[label])

print(label)

#read image

image = cv2.imread(image\_path)

#display an image window to show the image

cv2.imshow("Training on image...", cv2.resize(image, (400, 500)))

cv2.waitKey(1)

#face detect

face, rect = face\_detect(image)

if face is not None:

print(len(face))

print(rect)

faces.append(face)

labels.append(label)

cv2.destroyAllWindows()

cv2.waitKey(1)

cv2.destroyAllWindows()

return faces, labels

print("Reading data...")

faces,labels = reading\_training\_data('training-data')

print("Data read completed")

#print(faces)

#print(labels)

person\_names.append("Unknown")

#print(person\_names)

#print total faces and labels

print("Total faces: ", len(faces))

print("Total labels: ", len(labels))

#create our LBPH face recognizer

face\_recognizer = cv2.face.LBPHFaceRecognizer\_create()

#train our face recognizer of our training faces

face\_recognizer.train(faces, np.array(labels))

face\_recognizer.save("trainer.yml")

def predict(test\_img):

img = test\_img.copy()

face, rect = face\_detect(img)

print(face)

if face is not None:

label, confidence = face\_recognizer.predict(face)

print(label)

label\_text.append(person\_names[label])

for rect in rect:

draw\_rectangle(img, rect)

draw\_text(img, person\_names[label], rect[0], rect[1]-5)

return img

print("Predicting images...")

test\_img1 = cv2.imread("ssss.jpg")

#print(test\_img1)

predicted\_img1 = predict(test\_img1)

#print(label\_text)

found\_person = person\_names[-1];

if len(label\_text)>0:

cv2.imshow(found\_person, cv2.resize(predicted\_img1, (700,500)))

else:

cv2.imshow(found\_person, cv2.resize(predicted\_img1, (700,500)))

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.waitKey(1)

cv2.destroyAllWindows()