Installing OpenCv

The first thing to do is install the latest version of Python itself with Pip!





To ensure that opencv is installed or not, you can type in python in cmd to display it

in the CLI. Then add the module using import opencv if you do not have the warning

or error you can make sure that it is installed.

First step: Giving an image we want to know which part of the image, is the face,

Second step: then to generate the labels for the training data and then training our classifier

Third step: predicting the face

So for that we need to install two dependencies . one is OpenCv and second is numpy

import cv2 : OpenCV developers select this name while creating binding generators

import os: to create lable for our training data to handle later operations.

import numpy as np : to pass the training data to our classifier

We are going to define various functions in faceRecognition.py file, which are the common functionalities we will have one more file named tester.py and this will import faceRecognition.py file as a module and then we are going to test all these functions , using tester.py.

faceDetection(test\_img) :

faceDetection is a function that takes our test\_img (which has been already loaded using imread() of OpenCv.

gray\_img=cv2.cvtColor(test\_img,cv2.COLOR\_BGR2GRAY):

to convert the image to a gray image. Because our classifier takes a grey image . it doesn’t matter what color is our image while detecting a face , so we want to remove the distracting features. That’s why we convert the image to gray.

face\_haar\_cascade=cv2.CascadeClassifier('H:\\MLfaceDetection\\haarcascade\_frontalface\_default.xml'):

e define the face\_haar\_cascade variable. The classifier which we are going to use is called HaarClassifier. These classifiers are trained to detect certain kind of objects. So in our case we want a classifier which has been trained to detect faces. So have to download this classifier . can be downloaded from google. The file that we need is “haarcascade\_frontalface\_default.xml”. how haarcascade works is: they are trained to detect certain objects and then what we do is : we import different images and try to extract certain features of them. Haarcascade files are huge XML files and if you open them you will see different features are coded.

faces=face\_haar\_cascade.detectMultiScale(gray\_img,scaleFactor=1.3,minNeighbors=5):

this variable hass loaded face\_haar\_cascade classifier . Calling detectMultiScale will return us a rectangle around the detected face. We will pass the gray\_img and some other arguments such as: scaleFactor: If you select it bigger your detector work faster but detection rate will be less. Decreasing the size of image by %32. So while training this classifier, images which are bigger in size are likely to be not detected. So what we do is essentially trying to rescale the image, so it has more chances of getting detected by the classifier. And one more argument is minNeighbors :used for specifying how many neighbors each candidate rectangle should have to retain it. In case minNeighbors=0 there are lot of faces and lot of false positives will be detected. So when minNeighbors=5 , we are saying that the image should have at least 5 neighbors for it to be detected as a true positive.

return faces,gray\_img:

. Calling the faceDetection function will return us rectangle around the image and the gray img. Later in our classifier we will need the gray img

Then we have a tester.py file :

import faceRecognition as fr:

We are going to import faceDetection file as a module

test\_img=cv2.imread('H:\\ML faceDetection\\test\\Rishaw.jpeg'):

we load the test image using cv2.imread(). Then we give it an image.

faces\_detected,gray\_img= fr.faceDetection(test\_img):

we get faces\_detected and gray\_img by calling faceDetection module. We pass the test\_img which have been loaded using imread().

print("faces\_detected:" , faces\_detected):

printing the number of detected faces.

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

# cv2.rectangle(test\_img,(x,y),(x+w,y+h),(255,0,0),thickness=5):

We try to draw the rectangle around faces. We will give the test img . Because on that particular image we want to display the bounding boxes. So we are using the rectangle function from OpenCv. Here we will give the opposite diagonal points of the bounding boxes.

#cv2.imshow("test\_img", test\_img ):

Showing the image in bounding box with the caption “test\_img”.

Then in faceDetection file:

lebels\_for\_training\_data(directory):

faces=[]

faceID=[]

We define another function called lebels\_for\_training\_data to generate labels for each of the images in our training data. Because this haar classifier accepts images and the labels associated with it. That label should be only integers. In these training images we have mentioned 5 folders with ids 0 to 4. Each folder contains the pictures of one person. So later on we will create a dictionary with keyvalues with the name of each person. We will pass a directory as the argument of function. This function will go to the sub directories and will fetch all its images with their path, mention it with their labels.

for path,subdirnames,filenames in os.walk(directory):

os.walk generates the file names in a directory. This code will going to give us a path, a sub directory and a file name.

for filename in filenames:

if filename.startswith("."):

print( "skipping system file")

continue

id=os.path.basename(path)

#for each file we want to fetch the image path . we should not have the file name starting by “.”

Now we want to extract its id. (we want to extraxt the base name of the sub directories which are 0, 1, 2,3 , 4. )

img\_path=os.path.join(path,filename)

later on we will need the image path to feed it to our classifier.

print("img\_path", img\_path)

printing the path name for debugging purposes.

print("id",id)

test\_img=cv2.imread(img\_path)

if test\_img is None:

print("Image not loaded properly")

continue

if the image doesn’t get loaded properly with imread(), in that case it will going to return us “None”.

faces\_rect,gray\_img=faceDetection(test\_img)

if len(faces\_rect)!=1:

continue

Then we will call our faceDetection module again passing the test\_img as a parameter. This call will give us the gray image. Then we want to train our data. since we areassuming only single person images are using for classifier,our classifier is using only the single person images. We can not feed it , with same image containing two person ‘s images. So in case we have any images which is having multiple faces, we are going to skip it.

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

The rectangle will be returned by faces\_rect module

roi\_gray=gray\_img[y:y+w,x:x+h]

faces.append(roi\_gray)

faceID.append(int(id))

return faces,np.array(faceID)

so we want to extract the part of the image and since our classifier takes only the gray images, we are going to crop (x,y,w,h)=faces\_rect[0] this part from the gray image and feed it to our classifier. (roi=region of intreast). Means we are cropping the face part from the image and feed it to our classifier.

Our classifiers will take only the labels of type int , so we have to convert the id to int.

def train\_classifier(faces,faceID):

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

face\_recognizer.train(faces,faceID) # training the recognizer on the data that we generated.

return face\_recognizer

we define another function to train our classifier on this training imaged. So it is going to take faces,faceID, which is going to be returned by the lebels\_for\_training\_data function .( It will be the part of the gray image (which is face) and its label. )

Then we load our face\_recognizer variable inside the classifier to recognize our face. We are going to use LBPHFaceRecognizer\_ for this case. There are different face recognition techniques like fisher faces , LBPH(local binary pattern histogram ). How LBPS works? Instead of looking at the entire image we created the image into 3\*3 pixels . As we don’t know all the pixel values , we take the centeral value and then we compare it with its surrounding pixels. If the surrounding pixel value is lesser than the central value , we are going to assign that as zero, else other pixels will be assigned as one. And then we are going to take it in a clockwise menu and then we are going to find binary pattern. And then later we represent this as a decimal value for the central pixel . the face will be found in this way and then we are going to extract histogram of this face. Then whenever a new image has been given, we are going to create histogram which represents the image. Then we are going to give the image which is the nearest to this histogram.

def draw\_rect(test\_img,face):

(x,y,w,h)=face # extracting all the coordinates.

cv2.rectangle(test\_img,(x,y),(x+w,y+h),(255,0,0),thickness=5)

#drawing the bounding box around our face. It will take the test imag and the rectangle coordinates of the image.

def put\_text(test\_img,text,x,y):

cv2.putText(test\_img,text,(x,y),cv2.FONT\_HERSHEY\_DUPLEX,5,(255,0,0),6)

#writing the text that whose face it is. We call puttext function of opencv. font thickness=5, font size=6

Now , we go to our tester function and try to train our classifier for our training images. So in tester. Py file , we deactivated the part of code mentioned bellow.

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

# cv2.rectangle(test\_img,(x,y),(x+w,y+h),(255,0,0),thickness=5)

#resized\_img=cv2.resize\_img(test\_img,(1000,700))

#cv2.imshow("test\_img", test\_img )

#cv2.waitKey(1)

#cv2.destroyAllWindows

Now we want to call labels for training data from our tester function. As the labels for training data in faceRecognition module will return us faces and face id , so we add these part of code to our tester. Py file.

#faces,faceID= fr.lebels\_for\_training\_data("H:\\ML faceDetection\\Testimages")

It will take the directory where the trained images are.

#face\_regognizer= fr.train\_classifier(faces,faceID)

Then we pass it to our recognizer.

name={0:"Alwin", 1:"Mirdu",2:"Navarun",3:"Pulak",4:"Rishaw"}

we create a dictionary to tell which id belongs to which person.

for face in faces\_detected:

(x,y,w,h)=face # when we got one face detected

roi\_gray=gray\_img[y:y+w,x:x+h] # we are going to extract the face part from the gray image, that we got from the face detection, and now we extract the part of the image which is face.

label,confidence=face\_regognizer.predict(roi\_gray) # then we call predict function from opencv and we will give it the part of gray image , which is face. It will return us a label from 0 to 4 and confidence value, later we will use this confidence value to say at how much confidence we this value is predicted because it is going to predict in any values. So confidence value =0 means exact match. If confidence value usually is more than 35, 36 then we don’t want to predict that value because that is going to be wrong.

print("confidence: " ,confidence)

print("label",label)

fr.draw\_rect(test\_img,face)

predicted\_name=name[label] # it will give us a label and we want to extract the name out of that label.

if(confidence>37):

continue

fr.put\_text(test\_img,predicted\_name,x,y)

We can have multiple faces in our image , in a group photo in all the face detected we are taking the first face and we are taking its bounding coordinates

cv2.imshow("test\_img", test\_img )

cv2.waitKey(1)

cv2.destroyAllWindows

then when we run the tester.py file , it will start training our data.

After training we will directly save these data and load the classifier

So after training part in tester.py file, we add these codes:

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

face\_regognizer.read("H:\\ML faceDetection\\TrainedData\\trainingData.yml")

so next time we will not train it again we will just save the trained data and load it when we run tester.py file again.

After running the file once after saving the trained data, then we add this code:

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

face\_regognizer.read("H:\\ML faceDetection\\TrainedData\\trainingData.yml")