OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source BSD license.

Before starting you can read my article on [face detection](https://datascienceplus.com/face-and-eye-detection-with-opencv/) which will make this code more easy to understand.

For our face recognition model, we will have 3 phases:

 Prepare training data

 Train Face Recognizer

 Testing

In our dataset we will have two folders with faces of two persons. For example, in folder 1 we will have face images of person 1, and in other folder, face images of person 2. You can create your own personalised data or can download from my [Github Repo](https://github.com/arpit-dwivedi/Face-recognition/tree/master/face-recognition), where you can find the source code and other files.

**Coding**

As usually our first task will be to import required libraries.

#import OpenCV module

import cv2

#import os module for reading training data directories and paths

import os

#import numpy to convert python lists to numpy arrays as

#it is needed by OpenCV face recognizers

import numpy as np

Now preparing our training data-

#there is no label 0 in our training data so subject name for index/label 0 is empty

subjects = ["", "Arpit Dwivedi", "Udit Saxena"]

#function to detect face using OpenCV

def detect\_face(img):

#convert the test image to gray image as opencv face detector expects gray images

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

#load OpenCV face detector, I am using LBP which is fast

#there is also a more accurate but slow Haar classifier

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

#let's detect multiscale (some images may be closer to camera than others) images

#result is a list of faces

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5);

#if no faces are detected then return original img

if (len(faces) == 0):

return None, None

#under the assumption that there will be only one face,

#extract the face area

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

#return only the face part of the image

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

this function will read all persons’ training images, detect face from each image and will return two lists of exactly same size, one list of faces and another list of labels for each face.

def prepare\_training\_data(data\_folder\_path):

#------STEP-1--------

#get the directories (one directory for each subject) in data folder

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

#list to hold all subject faces

faces = []

#list to hold labels for all subjects

labels = []

#let's go through each directory and read images within it

for dir\_name in dirs:

#our subject directories start with letter 's' so

#ignore any non-relevant directories if any

if not dir\_name.startswith("s"):

continue;

#------STEP-2--------

#extract label number of subject from dir\_name

#format of dir name = slabel

#, so removing letter 's' from dir\_name will give us label

label = int(dir\_name.replace("s", ""))

#build path of directory containin images for current subject subject

#sample subject\_dir\_path = "training-data/s1"

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

#get the images names that are inside the given subject directory

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

#------STEP-3--------

#go through each image name, read image,

#detect face and add face to list of faces

for image\_name in subject\_images\_names:

#ignore system files like .DS\_Store

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

continue;

#build image path

#sample image path = training-data/s1/1.pgm

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

#read image

image = cv2.imread(image\_path)

#display an image window to show the image

cv2.imshow("Training on image...", image)

cv2.waitKey(100)

#detect face

face, rect = detect\_face(image)

#------STEP-4--------

#for the purpose of this tutorial

#we will ignore faces that are not detected

if face is not None:

#add face to list of faces

faces.append(face)

#add label for this face

labels.append(label)

cv2.destroyAllWindows()

cv2.waitKey(1)

cv2.destroyAllWindows()

return faces, labels

Let’s first prepare our training data data will be in two lists of same size one list will contain all the faces and other list will contain respective labels for each face.

print("Preparing data...")

faces, labels = prepare\_training\_data("training-data")

print("Data prepared")

#print total faces and labels

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

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

Now train the face recognizer-

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

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

Now predicting the images-  
Firstly we will have to draw rectangles around the detected face and the writing the text with it:

def draw\_rectangle(img, rect):

(x, y, w, h) = rect

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

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

cv2.putText(img, text, (x, y), cv2.FONT\_HERSHEY\_PLAIN, 1.5, (0, 255, 0), 2)

This function recognizes the person in image passed and draws a rectangle around detected face with name of the subject.

def predict(test\_img):

#make a copy of the image as we don't want to chang original image

img = test\_img.copy()

#detect face from the image

face, rect = detect\_face(img)

#predict the image using our face recognizer

label= face\_recognizer.predict(face)

#get name of respective label returned by face recognizer

label\_text = subjects[label[0]]

#draw a rectangle around face detected

draw\_rectangle(img, rect)

#draw name of predicted person

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

return img

And now the final part, giving the test image for recognition-

print("Predicting images...")

#load test images

test\_img1 = cv2.imread("test-data/1.jpg")

test\_img2 = cv2.imread("test-data/2.jpg")

#perform a prediction

predicted\_img1 = predict(test\_img1)

predicted\_img2 = predict(test\_img2)

print("Prediction complete")

#display both images

cv2.imshow(subjects[1], predicted\_img1)

cv2.imshow(subjects[2], predicted\_img2)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Conclusion**

We successfully made a model which can recognise the face from the input image, which we trained on custom dataset.