This project was done with this fantastic “Open Source Computer Vision Library”, the OpenCV. On this tutorial, we will be focusing on Raspberry Pi (so, Raspbian as OS) and Python, but I also tested the code on My windows and it also works fine. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. So, it’s perfect for real-time face recognition using a camera.

To create a complete project on Face Recognition, we must work on 3 very distinct phases:

1. Face Detection and Data Gathering
2. Train the Recognizer
3. Face Recognition

Once you have OpenCV installed in your System let’s test to confirm that your camera is working properly.

I am assuming that you have a WebCam already installed on your System.

Let’s start the first phase of our project. What we will do here, is starting from last step (Face Detecting), we will simply create a

dataset, where we will store for each id, a group of photos in gray with the portion that was used for face detecting.

Let's start the first phase of our project. What we will do here, is starting from last step (Face Detecting), we will simply create a dataset, where we will store for each id, a group of photos in gray with the portion that was used for face detecting.

Step 1 : Collecting Data

Let's start the first step of our project. What we will do here, is starting from last step (Face Detecting), we will simply create a dataset, where we will store for each id, a group of photos in gray with the portion that was used for face detecting.

Create a subdirectory where we will store our facial samples and name it "dataset":

```
import cv2
import os

cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video width
cam.set(4, 480) # set video height

face_detector =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# For each person, enter one numeric face id
face_id = input('\n enter user id end press <return> ==>  ')

print("\n [INFO] Initializing face capture. Look the camera and
wait ...")
# Initialize individual sampling face count
count = 0
```

```
while(True):
    ret, img = cam.read()
    img = cv2.flip(img, -1) # flip video image vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        count += 1

        # Save the captured image into the datasets folder
        cv2.imwrite("dataset/User." + str(face_id) + '.' +
str(count) + ".jpg", gray[y:y+h,x:x+w])

        cv2.imshow('image', img)

    k = cv2.waitKey(100) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break
    elif count >= 30: # Take 30 face sample and stop video
         break

# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

The code is very similar to the code that we saw for face detection. What we added, was an “input command” to capture a user id, that should be an integer number (1, 2, 3, etc)

```
face_id = input('\n enter user id end press  ==>  ')
```

And for each one of the captured frames, we should save it as a file on a “dataset” directory:

```
cv2.imwrite("dataset/User." + str(face_id) + '.' + str(count) +
".jpg", gray[y:y+h,x:x+w])
```

For example, for a user with a face_id = 1, the 4th sample file on dataset/ directory will be something like:

```
User.1.4.jpg
```

You must run the script each time that you want to aggregate a new user (or to change the photos for one that already exists).

Step 2 : Creating Trainer.yml

We must take all user data from our dataset and “trainer” the OpenCV Recognizer. This is done directly by a specific OpenCV function. The result will be a .yml file that will be saved on a “trainer/” directory.

```
import cv2
import numpy as np
from PIL import Image
import os

# Path for face image database
path = 'dataset'

recognizer = cv2.face.LBPHFaceRecognizer_create()
detector =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml");

# function to get the images and label data
def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    faceSamples=[]
    ids = []
    for imagePath in imagePaths:
        PIL_img = Image.open(imagePath).convert('L') # convert it
to grayscale
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)
```

```
        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)
    return faceSamples,ids

print ("\n [INFO] Training faces. It will take a few seconds. Wait
...")
faces,ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))

# Save the model into trainer/trainer.yml
recognizer.write('trainer/trainer.yml') # recognizer.save() worked
on Mac, but not on Pi

# Print the numer of faces trained and end program
print("\n [INFO] {0} faces trained. Exiting
Program".format(len(np.unique(ids))))
```

We will use as a recognizer, the LBPH (LOCAL BINARY PATTERNS HISTOGRAMS) Face Recognizer, included on OpenCV package. We do this in the following line:

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
```

The function “getImagesAndLabels (path)”, will take all photos on directory: “dataset/”, returning 2 arrays: “Ids” and “faces”. With those arrays as input, we will “train our recognizer”.

As a result, a file named “trainer.yml” will be saved in the trainer directory that was previously created by us.

Step 3 : Recognizer

We will capture a fresh face on our camera and if this person had his face captured and trained before, our recognizer will make a “prediction” returning its id and an index, shown how confident the recognizer is with this match.

```
import cv2
import numpy as np
import os

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);

font = cv2.FONT_HERSHEY_SIMPLEX

#iniciate id counter
id = 0

# names related to ids: example ==> Marcelo: id=1,  etc
names = ['None', 'Marcelo', 'Paula', 'Ilza', 'Z', 'W']

# Initialize and start realtime video capture
cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video widht
cam.set(4, 480) # set video height

# Define min window size to be recognized as a face
minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)

while True:
    ret, img =cam.read()
    img = cv2.flip(img, -1) # Flip vertically
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(minW), int(minH)),
       )

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

```
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
        id, confidence = recognizer.predict(gray[y:y+h,x:x+w])

        # Check if confidence is less them 100 ==> "0" is match
        if (confidence < 100):
            id = names[id]
            confidence = "  {0}%".format(round(100 - confidence))
        else:
            id = "unknown"
            confidence = "  {0}%".format(round(100 - confidence))

        cv2.putText(img, str(id), (x+5,y-5), font, 1,
(255,255,255), 2)
        cv2.putText(img, str(confidence), (x+5,y+h-5), font, 1,
(255,255,0), 1)

    cv2.imshow('camera',img)

    k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break

# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

We are including here a new array, so we will display “names”, instead of numbered ids:

```
names = ['None', 'Devi', 'Gopinath', 'Jayalalitha', 'Z', 'W']
```

So, for example: Devi will the user with id = 1; Gopinath: id=2, etc.

Next, we will detect a face, same we did before with the haarCascade classifier. Having a detected face we can call the most important function in the above code:

```
id, confidence = recognizer.predict(gray portion of the face)
```

The recognizer.predict (), will take as a parameter a captured portion of the face to be analyzed and will return its probable owner, indicating its id and how much confidence the recognizer is in relation with this match.

Note that the confidence index will return “zero” if it will be cosidered a perfect match

And at last, if the recognizer could predict a face, we put a text over the image with the probable id and how much is the “probability” in % that the match is correct (“probability” = 100 — confidence index). If not, an “unknow” label is put on the face.