Real-Time Face Recognition: An End-to-End Project

Contents

Vision recognition and machine learning

This project is covering an End to End process from start to end with the different steps. It needs to be highlighted this project is not based on the latest or the best algorithms/methodology. It should be seen and used as a taster to what can be achieved with fairly simple steps. At a later stage this can be enhanced to cover more advanced aspects as SVM (Simple Vector Machine), DNN (Deep Neural Network) and CNN (Convoluted Neural Network).

OpenCV

OpenCV is the most popular library for computer vision. Originally written in C/C++, it now provides bindings for Python.

OpenCV uses machine learning algorithms to search for faces within a picture. Because faces are so complicated, there isn't one simple test that will tell you if it found a face or not. Instead, there are thousands of small patterns and features that must be matched. The algorithms break the task of identifying the face into thousands of smaller, bite-sized tasks, each of which is easy to solve. These tasks are also called classifiers.

For something like a face, you might have 6,000 or more classifiers, all of which must match for a face to be detected (within error limits, of course). But therein lies the problem: for face detection, the algorithm starts at the top left of a picture and moves down across small blocks of data, looking at each block, constantly asking, "Is this a face? … Is this a face? … Is this a face?" Since there are 6,000 or more tests per block, you might have millions of calculations to do, which will grind your computer to a halt.

To get around this, OpenCV uses cascades. What's a cascade? The best answer can be found in the dictionary: "a waterfall or series of waterfalls."

Like a series of waterfalls, the OpenCV cascade breaks the problem of detecting faces into multiple stages. For each block, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30 to 50 of these stages or cascades, and it will only detect a face if all stages pass.

The advantage is that the majority of the picture will return a negative during the first few stages, which means the algorithm won't waste time testing all 6,000 features on it. Instead of taking hours, face detection can now be done in real time.

Cascades in Practice

Though the theory may sound complicated, in practice it is quite easy. The cascades themselves are just a bunch of XML files that contain OpenCV data used to detect objects. You initialize your code with the cascade you want, and then it does the work for you.

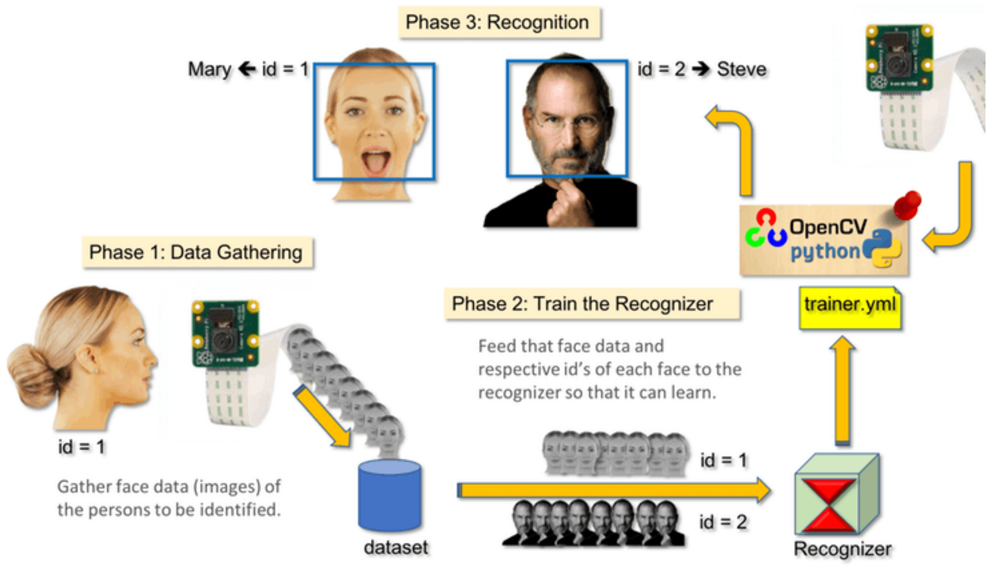
Since face detection is such a common case, OpenCV comes with a number of built-in cascades for detecting everything from faces to eyes to hands to legs. There are even cascades for non-human things. For example, if you run a banana shop and want to track people stealing bananas, this guy has built one for that!

3 Phases

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

- Face Detection and Data Gathering
- Train the Recognizer
- Face Recognition

The below block diagram resumes those phases:


Phase 3: Recognition
Mary ← id = 1
id = 2 → Steve
OpenCV python
Phase 1: Data Gathering
id = 1
Gather face data (images) of the persons to be identified.
dataset
Phase 2: Train the Recognizer
Feed that face data and respective id's of each face to the recognizer so that it can learn.
trainer.yml
id = 1
id = 2
Recognizer

Prerequisite

Ensure you have python 3.6 or higher installed.

To verify open up your Python interpreter:

IDLE, PyCharm, MU and confirm that you are running the 3.6 (or above) version.

Crate a directory/project/venv to your own liking example: "**FacialRecognitionProjec**t"

Step 1: Installing OpenCV 4 Package

Until fairly recently installing OpenCV has been anything then plain sailing. Anyhow things have progress and this step is now straight forward. Depending on your IDE and environment this can be done via PIP or build in package manager

PIP

If PIP is used please use the following from the command line:

- "Pip install opencv-contrib-python"

The above line works with RPI and vanilla python3.

PyCharm

- Crate a new project
- Click on file and go to Settings
- Identify Project:"your crated project" and click on the Project Interpreter
- Click on the plus sign (on the right hand side)
- At the magnification glass type "opencv-contrib-python"

Now, open up your Python interpreter:
Inside the interpreter (the ">>>" will appear), import the OpenCV library:

```
import cv2
```

If no error messages appear, the OpenCV is correctly installed
You can also check the OpenCV version installed:

```
cv2.__version__
```

The 4.x.x should appear

Step 2: Testing Your Camera

Once you have OpenCV installed let's test to confirm that your camera is working properly.
Enter the below Python code on your IDE:
Camera_test.py

```
import cv2

cap = cv2.VideoCapture(1) # default is 0, my front camera is 1

while(True):
    ret, frame = cap.read()
    # frame = cv2.flip(frame, 1) # Flip camera vertically
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame', frame)
    cv2.imshow('gray', gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

The above code will capture the video stream that will be generated by your WEB Cam, displaying both, in BGR colour and Grey mode.
To finish the program, you must press the key [ESC] on your keyboard. Click your mouse on the video window, before pressing [ESC].

Step 3: Face Detection

The most basic task on Face Recognition is of course, "Face Detecting". Before anything, you must "capture" a face (Phase 1) in order to recognise it, when compared with a new face captured on future (Phase 3).The most common way to detect a face (or any objects), is using the "Haar Cascade classifier" Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed

by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. The good news is that OpenCV comes with a trainer as well as a detector. If you want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one. Its full details are given here: Cascade Classifier Training.
If you do not want to create your own classifier, OpenCV already contains many pre-trained classifiers for face, eyes, smile, etc. Those XML files can be found under "Lib/site-packages/cv2/data/"
Enough theory, let's create a face detector with OpenCV!

faceDetection.py

```
import numpy as np
import cv2

faceCascade = cv2.CascadeClassifier('Lib/site-packages/cv2/data/haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(1)
#cap.set(3,640) # set Width
#cap.set(4,480) # set Height
while True:
    ret, img = cap.read()
    # img = cv2.flip(img, -1)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]


    cv2.imshow('video',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        break
cap.release()
cv2.destroyAllWindows()
```

Believe it or not, the above few lines of code are all you need to detect a face, using Python and OpenCV.
When you compare with the last code used to test the camera, you will realize that few parts were added to it. Note the line below:
faceCascade = cv2.CascadeClassifier('**Your path**/haarcascade_frontalface_default.xml')
This is the line that loads the "classifier" (that must be under your project directory with the **correct path**).
Then, we will set our camera and inside the loop, load our input video in grayscale mode (same we saw before).
Now we must call our classifier function, passing it some very important parameters, as scale factor, number of neighbors and minimum size of the detected face.

```
faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
```

Where,

- **gray** is the input grayscale image.
- **scaleFactor** is the parameter specifying how much the image size is reduced at each image scale. It is used to create the scale pyramid.
- **minNeighbors** is a parameter specifying how many neighbors each candidate rectangle should have, to retain it. A higher number gives lower false positives.
- **minSize** is the minimum rectangle size to be considered a face.

The function will detect faces on the image. Next, we must "mark" the faces in the image, using, for example, a blue rectangle. This is done with this portion of the code:

```
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
```

If faces are found, it returns the positions of detected faces as a rectangle with the left up corner (x,y) and having "w" as its Width and "h" as its Height ==> (x,y,w,h).
Once we get these locations, we can create an "ROI" (drawn rectangle) for the face and present the result with *imshow()* function.
Run the above python Script on your python environment,

```
python faceDetection.py
```

You can also include classifiers for "eyes detection" or even "smile detection". On those cases, you will include the classifier function and rectangle draw inside the face loop, because would be no sense to detect an eye or a smile outside of a face.

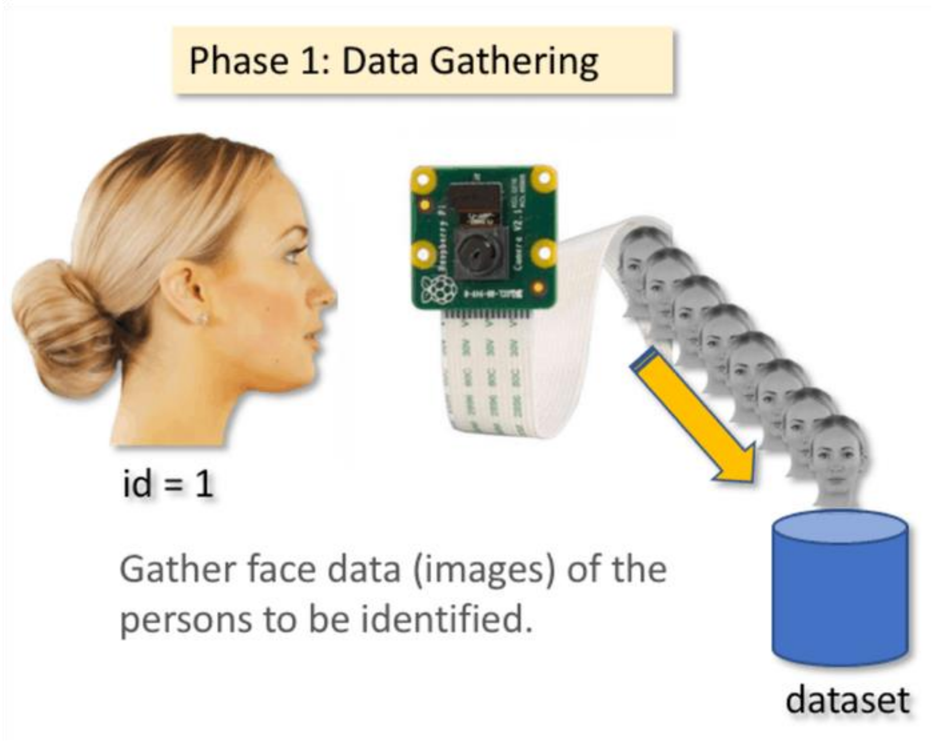
Examples

On Coder Dojo GitHub you will find other examples:

- faceEyeDetection.py
- faceSmileDetection.py
- faceSmileEyeDetection.py

Step 4: Data Gathering

FACE RECOGNITION - 3 parts
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 grey with the portion that was used for face detecting.



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

```
mkdir dataset
```

Face_dataset.py

```python
import cv2
import os
cam = cv2.VideoCapture(1)
cam.set(3, 640) # set video width
cam.set(4, 480) # set video height
face_detector = cv2.CascadeClassifier('Lib/site-packages/cv2/data/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  ==>  ')
```

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])
```

Note that for saving the above file, you must have imported the library "**os**". Each file's name will follow the structure:

```
User.face_id.count.jpg
```

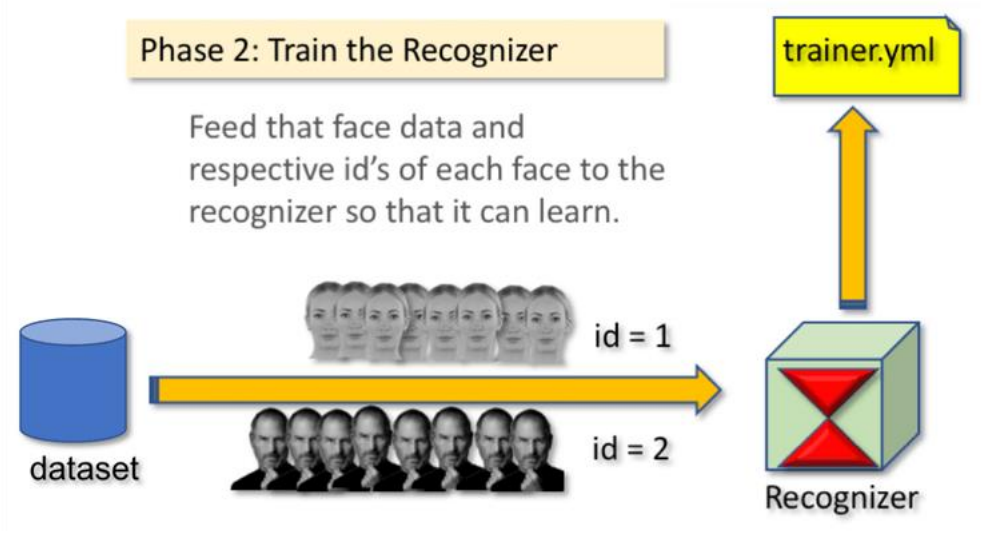
For example, for a user with a face_id = 1, the 4th sample file on dataset/ directory will be something like:

```
User.1.4.jpg
```

In the example script “face_dataset.py” I’m capturing 30 samples from each id. You can change it on the last "elif". The number of samples is used to break the loop where the face samples are captured.
Run the Python script and capture a few Ids. 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 5: Trainer

On this second phase, 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.


Phase 2: Train the Recognizer
Feed that face data and respective id's of each face to the recognizer so that it can learn.
trainer.yml
id = 1
id = 2
dataset
Recognizer

So, let's start creating a subdirectory where we will store the trained data:

```
mkdir trainer
```

Confirm if you have the PIL library installed. If not, run the below command in Terminal:
pip install pillow or for PyCharm add it in the same way as for the openCV package.

```
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("Lib/site-packages/cv2/data/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 Recogniser, included on OpenCV package. We do this in the following line:

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

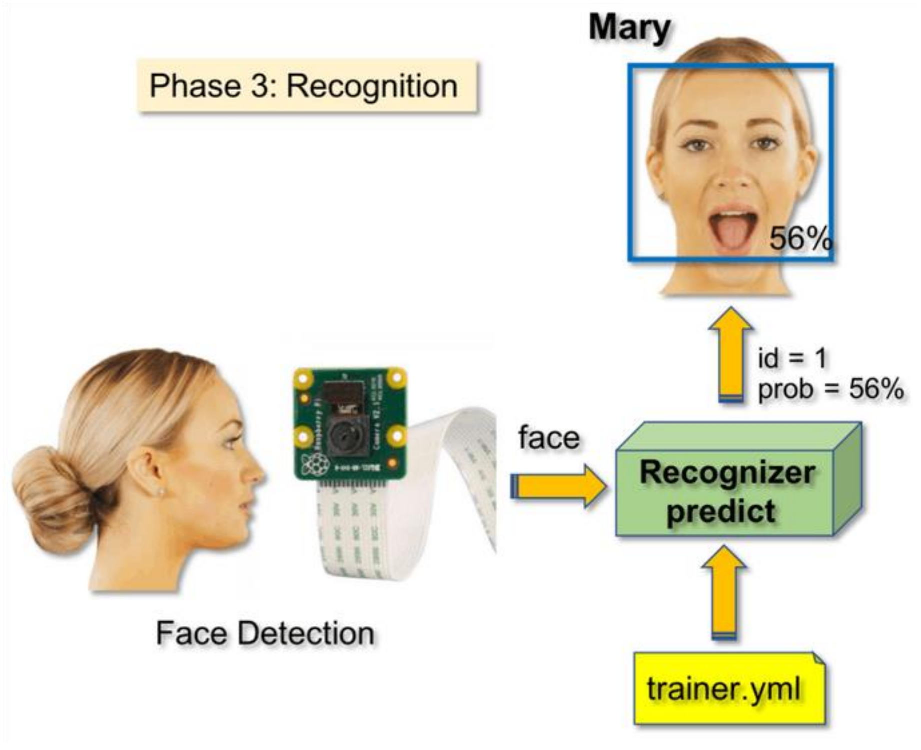
The function "getImagesAndLabels (path)", will take all photos on directory: "dataset/", returning 2 arrays: "Ids" and "faces". With those arrays as input, we will "train our recognizer":

```
recognizer.train(faces, ids)
```

As a result, a file named "trainer.yml" will be saved in the trainer directory that was previously created by us.
That's it! I included the last print statement where I displayed for confirmation, the number of User's faces we have trained.
Every time that you perform Phase 1, Phase 2 must also be run.

Step 6: Recognizer

Now, we reached the final phase of our project. Here, 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.



Face_recognition.py

```
import cv2


recogniser = cv2.face.LBPHFaceRecognizer_create()
recogniser.read('venv/trainer/trainer.yml')
cascadePath = "venv/Lib/site-packages/cv2/data/haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX
# id counter
id = 0
# names related to ids: example ==> Kajsa: id=2,  etc
names = ['None', 'Per', 'kajsa', 'Kiyono', 'Z', 'W']
# Initialize and start realtime video capture
cam = cv2.VideoCapture(1)
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))
        # minSize=(20, 20)
    )
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        id, confidence = recogniser.predict(gray[y:y + h, x:x + w])
        # Check if confidence is less then 100 ==> "0" is perfect 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', ‘Per', 'Kajsa', 'Kiyono', 'Z', 'W']
```

So, for example: Kajsa will the user with id = 2; Kiyono: id=3, etc.
Next, we will detect a face, same we did before with the haasCascade 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 analysed 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.
Below a gif with the result:

Credits

- Shantnu Tiwari
- Adrian Rosebrock,
- Ramiz Raja
- Anirban Kar
- Marcelo @ MJRoBot.org