MINI PROJECT – I

(2019-20)

Multi Face Recognition

Final Report

Institute of Engineering & Technology

Team Members

Saurabh Chhonkar
(171500299)
Shubham Singh
(171500335)
Prabhat Kumar
(171500219)

Supervised By

Mr. Piyush vashishtha

Asst. Professor

Department of Computer Engineering & Applications

ACKNOWLEDGEMENT

I thank the almighty for giving us the courage and perseverance in completing the main project. This project itself is acknowledgements for all those people who have given us their heartfelt co-operation in making this project a success.

With extreme jubilance and deepest gratitude, I would like to thank Head of the CSE Department, Prof. Anand Singh Jalal for his constant encouragement.

I am greatly indebted to project guide Mr. Piyush Vashistha, Assistant Professor of computer engineering department, for providing valuable guidance at every stage of this project work. I am profoundly grateful towards the unmatched services rendered by him.

Our special thanks to all the faculty Computer Engineering department and peers for their valuable advises at every stage of this work. Last but not least , we would like to express our deep sense of gratitude and earnest thanks giving to our dear parents for their moral support and heartfelt cooperation in doing the main project.

ABSTRACT

The face is one of the easiest ways to distinguish the individual identity of each other. Face recognition is a personal identification system that uses personal characteristics of a person to identify the person's identity. Human face recognition procedure basically consists of two phases, namely face detection, where this process takes place very rapidly in humans, except under conditions where the object is located at a short distance away, the next is the introduction, which recognize a face as individuals. Stage is then replicated and developed as a model for facial image recognition (face recognition) is one of the much-studied biometrics technology and developed by experts. There are two kinds of methods that are currently popular in developed face recognition pattern namely, Eigenface method and Fisherface method. We are using LBPH (Local Binary Patterns Histograms) classifier to recognize the faces from the images. It compares neighbouring pixels of a pixel and creates a histogram out of it for comparing faces.

CONTENT

1. Introduction

The goal of this article is to provide an easier human-machine interaction routine when user authentication is needed through face detection and recognition. With the aid of a regular web camera, a machine is able to detect and recognize a person's face; a custom login screen with the ability to filter user access based on the users' facial features will be developed. The objectives of this thesis are to provide a set of detection algorithms that can be later packaged in an easily portable framework amongst the different processor architectures we see in machines (computers) today. These algorithms must provide at least a 95% successful recognition rate, out of which less than 3% of the detected faces are false positives.

1.1 General Introduction to the topic:

The following document is a report on the mini project on Multi face recognition. It involved building a system for face detection and face recognition using several classifiers available in the open computer vision library(OpenCV). Face recognition is a non-invasive identification system and faster than other systems since multiple faces can be analysed at the same time.
The difference between face detection and identification is, face detection is to identify a face from an image and locate the face. Face recognition is making the decision ”whose face is it ? ”, using an image database. In this project both are accomplished using different techniques and are described below.

1.2 Hardware and Software requirement:

Hardware requirement:

- Intel processor or any other processor
- 4GB Ram and 100GB disk space
- Web camera

Software requirement:

- Anaconda and spyder
- Python 3.7

2.OpenCV-Python

OpenCV was started at Intel in the year 1999 by **Gary Bradsky**. The first release came a little later in the year 2000. OpenCV essentially stands for **Open Source Computer Vision Library**. Although it is written in optimized C/C++, it has interfaces for Python and Java along with C++. OpenCV boasts of an active user base all over the world with its use increasing day by day due to the surge in computer vision applications.

OpenCV-Python is the python API for OpenCV. You can think of it as a python wrapper around the C++ implementation of OpenCV. OpenCV-Python is not only fast (since the background consists of code written in C/C++) but is also easy to code and deploy(due to the Python wrapper in foreground). This makes it a great choice to perform computationally intensive programs.

Installation

OpenCV-Python supports all the leading platforms like Mac OS, Linux, and Windows. It can be installed in either of the following ways:

Unofficial pre-built OpenCV packages for Python.

Packages for standard desktop environments (Windows, macOS, almost any GNU/Linux distribution)

- run `pip install opencv-python` if you need only main modules

- run pip install opencv-contrib-python if you need both main and contrib modules

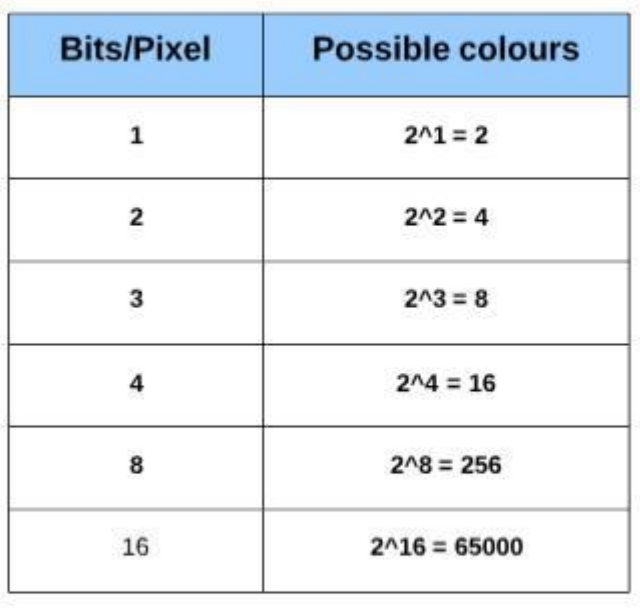
You can either use Jupyter notebooks or any Python IDE of your choice for writing the scripts.

3. Images as Arrays

An image is nothing but a standard Numpy array containing pixels of data points. More the number of pixels in an image, the better is its resolution. You can think of pixels to be tiny blocks of information arranged in the form of a 2 D grid, and the depth of a pixel refers to the color information present in it. In order to be processed by a computer, an image needs to be converted into a binary form. The color of an image can be calculated as follows:

```
Number of colors/ shades = 2^bpp where bpp represents bits per pixel.
```

Naturally, more the number of bits/pixels, more possible colors in the images. The following table shows the relationship more clearly.

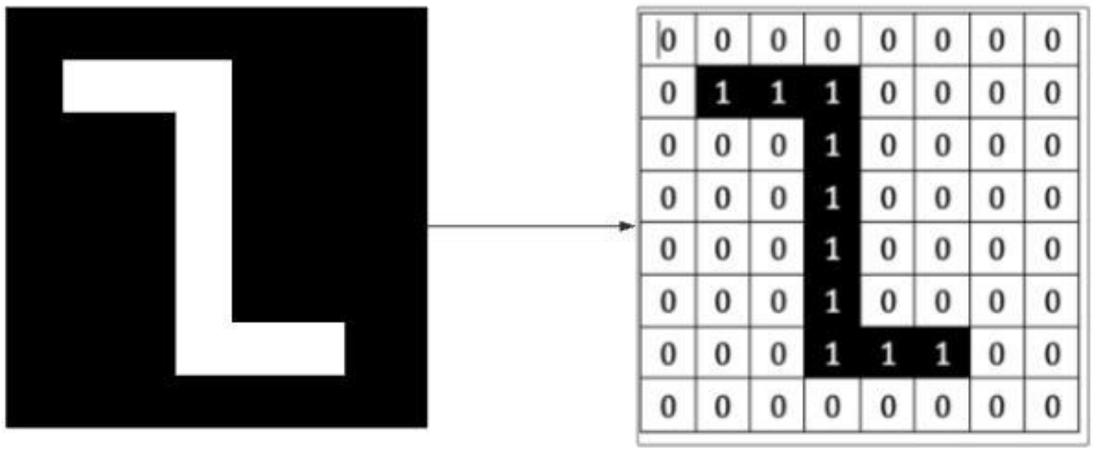


Bits/Pixel	Possible colours
1	2^1 = 2
2	2^2 = 4
3	2^3 = 8
4	2^4 = 16
8	2^8 = 256
16	2^16 = 65000

Let us now have a look at the representation of the different kinds of images:

1. Binary Image

A binary image consists of 1 bit/pixel and so can have only two possible colors, i.e., black or white. Black is represented by the value 0 while 1 represents white.

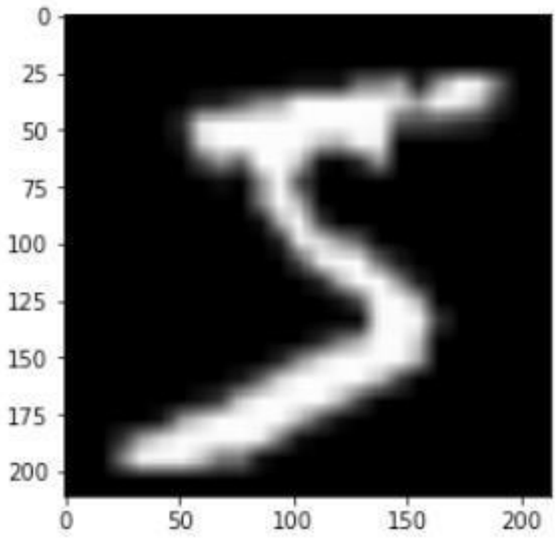


0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0

Representation of a black and white image in form of a binary where '1' represents pure white while '0' represents black. Here the image is represented by 1 bit/pixel which means image can be represented by only 2 possible colours since 2^1= 2

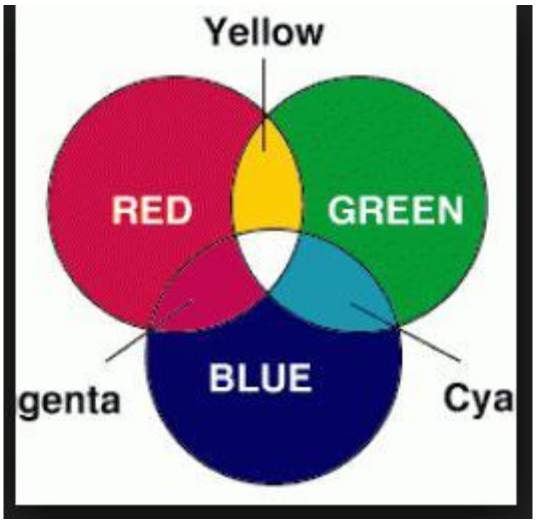
2. Grayscale image

A grayscale image consists of 8 bits per pixel. This means it can have 256 different shades where 0 pixels will represent black color while 255 denotes white. For example, the image below shows a grayscale image represented in the form of an array. A grayscale image has only 1 channel where the channel represents dimension.



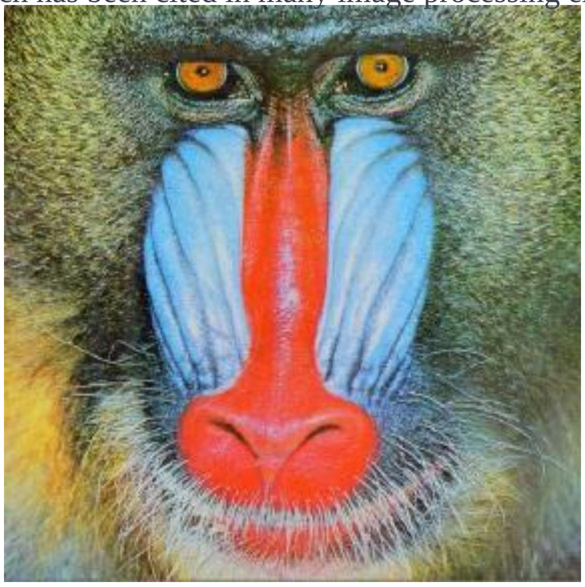
3. Colored image

Colored images are represented as a combination of Red, Blue, and Green, and all the other colors can be achieved by mixing these primary colors in correct proportions.



source

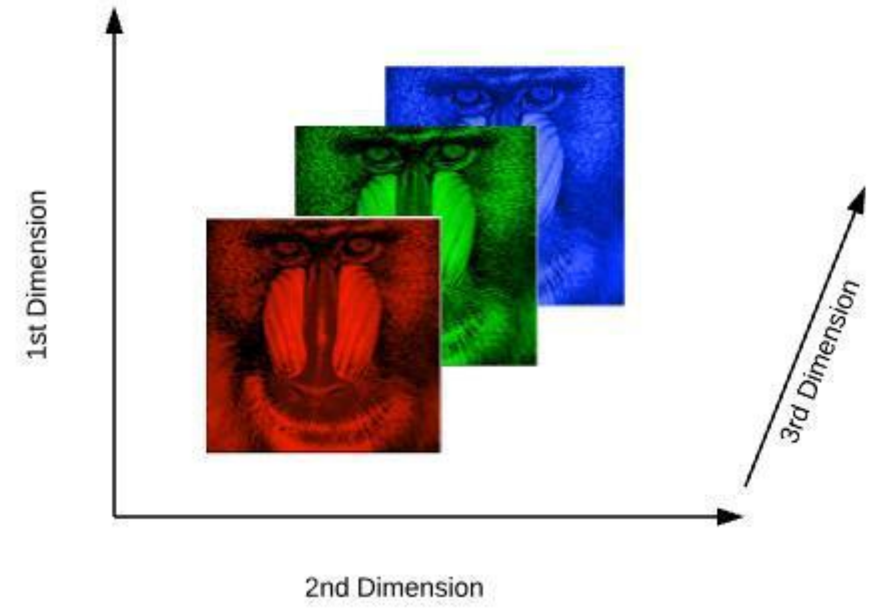
A colored image also consists of 8 bits per pixel. As a result, 256 different shades of colors can be represented with 0 denoting black and 255 white. Let us look at the famous colored image of a mandrill which has been cited in many image processing examples.



If we were to check the shape of the image above, we would get:

```
Shape

(288, 288, 3)

288: Pixel width

288: Pixel height

3: color channel
```

This means we can represent the above image in the form of a three-dimensional array.



4. Images and OpenCV

Before we jump into the process of face detection, let us learn some basics about working with OpenCV. In this section we will perform simple operations on images using OpenCV like opening images, drawing simple shapes on images and interacting with images through call backs. This is necessary to create a foundation before we move towards the advanced stuff.

Importing Images in OpenCV

Using Jupyter notebooks

Steps:

- **Import the necessary libraries**

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
```

- Read in the image using the **imread** function. We will be using the colored 'mandrill' image for demonstration purpose. It can be downloaded from here

```
img_raw = cv2.imread('image.jpg')
```

- **The type and shape of the array.**

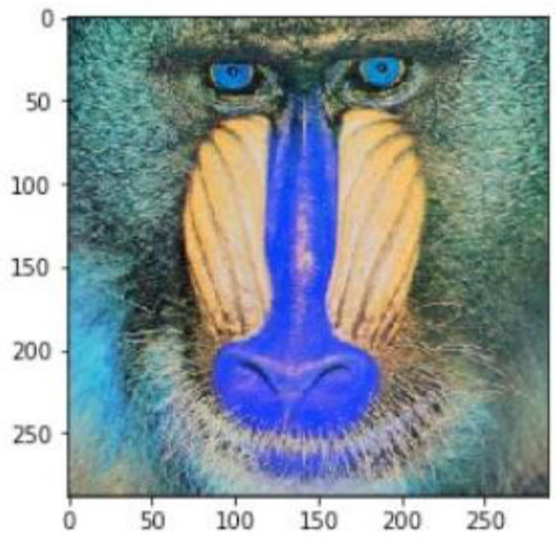
```
type(img_raw)
numpy.ndarray
```

```
img_raw.shape
(1300, 1950, 3)
```

Thus, the .png image gets transformed into a numpy array with a shape of 1300x1950 and has 3 channels.

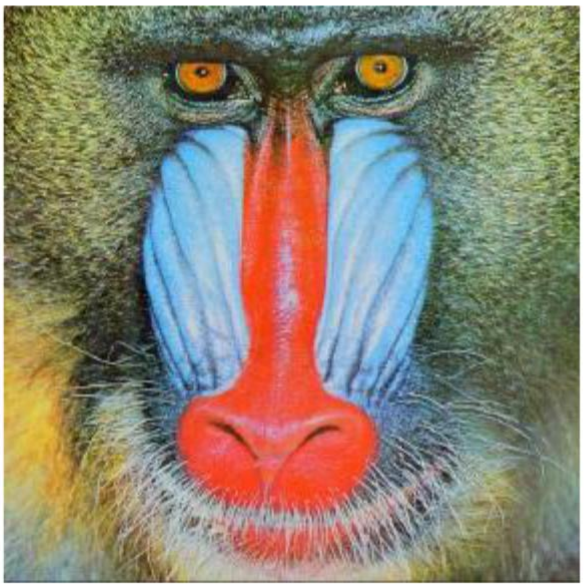
- **viewing the image**

```
plt.imshow(img_raw)
```



What we get as an output is a bit different concerning color. We expected a bright colored image but what we obtain is an image with some bluish tinge. That happens because OpenCV and matplotlib have different orders of primary colors. Whereas OpenCV reads images in the form of BGR, matplotlib, on the other hand, follows the order of RGB. Thus, when we read a file through OpenCV, we read it as if it contains channels in the order of blue, green and red. However, when we display the image using matplotlib, the red and blue channel gets swapped and hence the blue tinge. To avoid this issue, we will transform the channel to how matplotlib expects it to be using the cvtColor function.

```
img = cv2.cvtColor(img_raw, cv2.COLOR_BGR2RGB)

plt.imshow(img_rgb)
```



Using Python Scripts

Jupyter Notebooks are great for learning, but when dealing with complex images and videos, we need to display them in their own separate windows. In this section, we will be executing the code as a .py file. You can use Pycharm, Sublime or any IDE of your choice to run the script below.

```
import cv2

img = cv2.imread('image.jpg')

while True:

    cv2.imshow('mandrill',img)
    if cv2.waitKey(1) & 0xFF == 27:
        break



cv2.destroyAllWindows()
```

In this code, we have a condition, and the image will only be shown if the condition is true. Also, to break the loop, we will have two conditions to fulfill:

- The cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues.
- The second condition pertains to the pressing of the Escape key on the keyboard. Thus, if 1 millisecond has passed and the escape key is pressed, the loop will break and program stops.
- cv2.destroyAllWindows() simply destroys all the windows we created. If you want to destroy any specific window, use the function **cv2.destroyWindow()** where you pass the exact window name as the argument.

5.Savings images

The images can be saved in the working directory as follows:

```
cv2.imwrite('final_image.png',img)
```

Where the final_image is the name of the image to be saved.

Using Python Scripts

Jupyter Notebooks are great for learning, but when dealing with complex images and videos, we need to display them in their own separate windows. In this section, we will be executing the code as a .py file. You can use Pycharm, Sublime or any IDE of your choice to run the script below.

```
import cv2
```

```
img = cv2.imread('image.jpg')
while True:
    cv2.imshow('mandrill',img)

    if cv2.waitKey(1) & 0xFF == 27:
        break


cv2.destroyAllWindows()
```

In this code, we have a condition, and the image will only be shown if the condition is true. Also, to break the loop, we will have two conditions to fulfill:

- The cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues.
- The second condition pertains to the pressing of the Escape key on the keyboard. Thus, if 1 millisecond has passed and the escape key is pressed, the loop will break and program stops.
- cv2.destroyAllWindows() simply destroys all the windows we created. If you want to destroy any specific window, use the function **cv2.destroyWindow()** where you pass the exact window name as the argume.

6.Basic Operations on Images

In this section, we will learn how we can draw various shapes on an existing image to get a flavour of working with OpenCV.

Drawing on Images

- Begin by importing necessary libraries.

```
import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import cv2
```

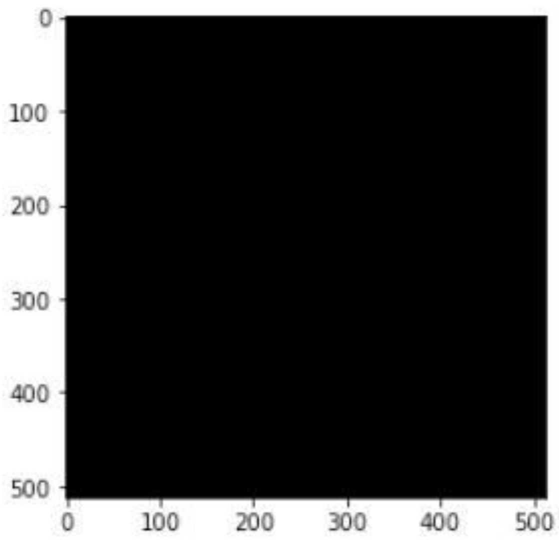
- Create a black image which will act as a template.

```
image_blank = np.zeros(shape=(512,512,3),dtype=np.int16)
```

- Display the black image.

```
plt.imshow(image_blank)
```

```
<matplotlib.image.AxesImage at 0x123141160>
```



Function & Attributes

The generalised function for drawing shapes on images is:

```
cv2.shape(line, rectangle etc)(image,Pt1,Pt2,color,thickness)
```

There are some common arguments which are passed in function to draw shapes on images:

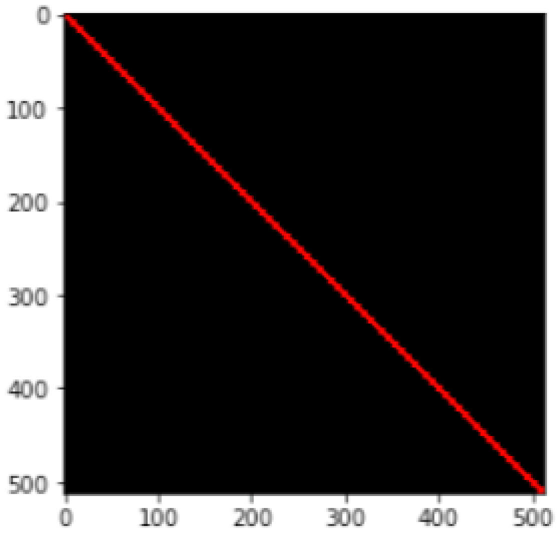
- Image on which shapes are to be drawn
- co-ordinates of the shape to be drawn from Pt1(top left) to Pt2(bottom right)

- **Colour**: The colour of the shape that is to be drawn. It is passed as a tuple, eg: (255,0,0). For grayscale, it will be the scale of brightness.
- The thickness of the geometrical figure.

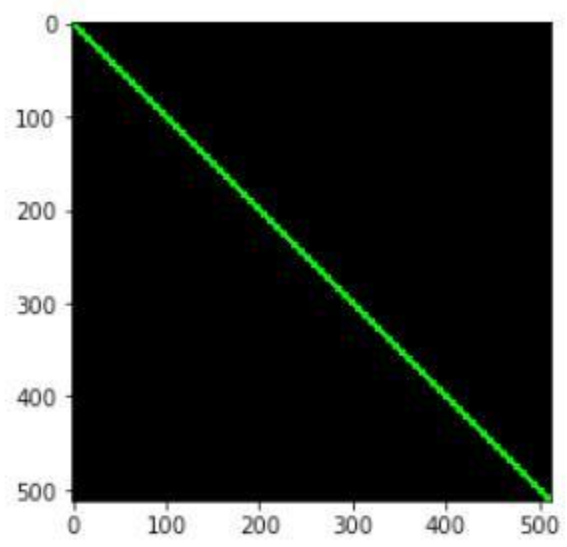
1. Straight Line

Drawing a straight line across an image requires specifying the points, through which the line will pass.

```
# Draw a diagonal red line with thickness of 5 px

line_red = cv2.line(img,(0,0),(511,511),(255,0,0),5)

plt.imshow(line_red)
```



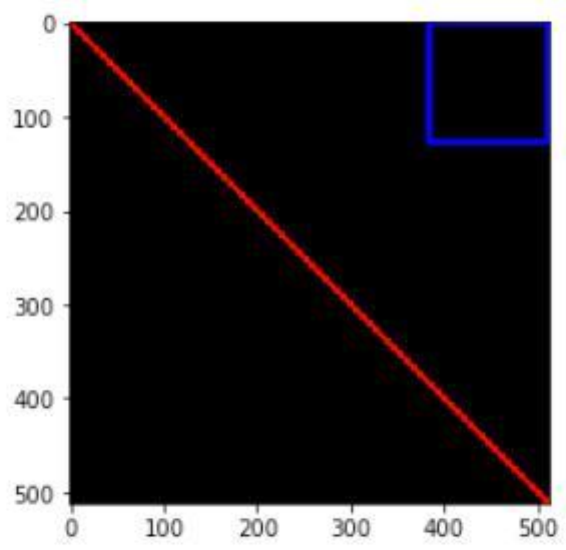
```
# Draw a diagonal green line with thickness of 5 px

line_green = cv2.line(img,(0,0),(511,511),(0,255,0),5)

plt.imshow(line_green)
```



2. Rectangle

For a rectangle, we need to specify the top left and the bottom right coordinates.

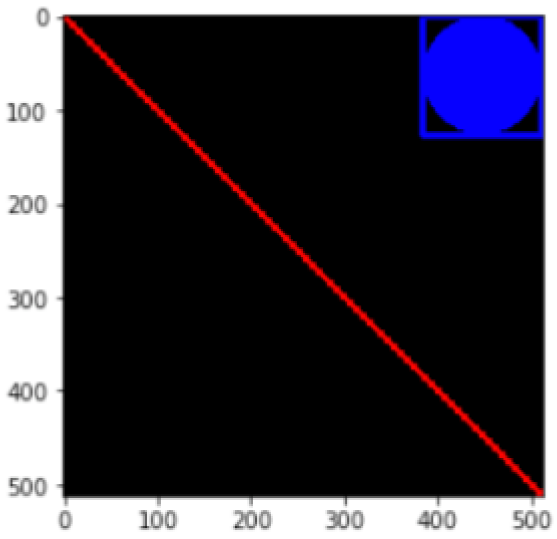
```
#Draw a blue rectangle with a thickness of 5 px


rectangle= cv2.rectangle(img,(384,0),(510,128),(0,0,255),5)

plt.imshow(rectangle)
```



3. Circle

For a circle, we need to pass its center coordinates and radius value. Let us draw a circle inside the rectangle drawn above

```
img = cv2.circle(img,(447,63), 63, (0,0,255), -1) # -1 corresponds to a
filled circle
plt.imshow(circle)
```

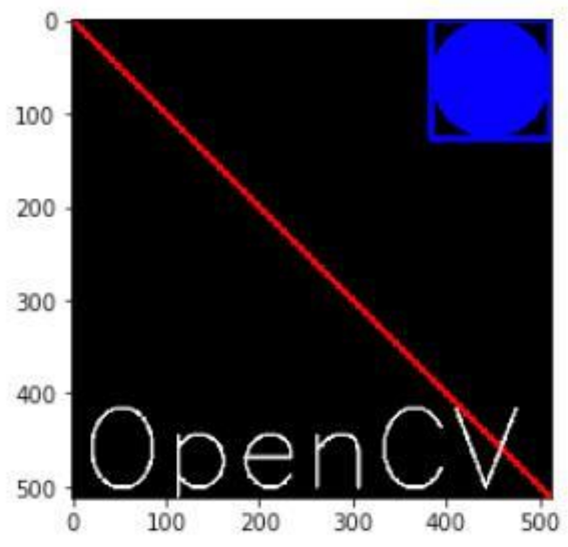


Writing on Images

Adding text to images is also similar to drawing shapes on them. But you need to specify certain arguments before doing so:

- Text to be written
- coordinates of the text. The text on an image begins from the bottom left direction.
- Font type and scale.
- Other attributes like colour, thickness and line type. Normally the line type that is used is lineType = cv2.LINE_AA.

```
font = cv2.FONT_HERSHEY_SIMPLEX
text = cv2.putText(img,'OpenCV',(10,500), font,
4,(255,255,255),2,cv2.LINE_AA)
plt.imshow(text)
```



These were the minor operations that can be done on images using OpenCV. Feel free to experiment with the shapes and text.

7.Face Detection

Overview

Face detection is a technique that identifies or locates human faces in digital images. A typical example of face detection occurs when we take photographs through our smartphones, and it instantly detects faces in the picture. Face detection is different from Face recognition. Face detection detects merely the presence of faces in an image while facial recognition involves identifying whose face it is. In this article, we shall only be dealing with the former.

Face detection is performed by using classifiers. A classifier is essentially an algorithm that decides whether a given image is positive(face) or negative(not a face). A classifier needs to be trained on thousands of images with and without faces. Fortunately, OpenCV already has two pre-trained face detection classifiers, which can readily be used in a program. The two classifiers are:

- Haar Classifier and
- Local Binary Pattern(LBP) classifier.

In this article, however, we will only discuss the Haar Classifier.

Haar feature-based cascade classifiers

Haar-like features are digital image features used in object recognition. They owe their name to their intuitive similarity with Haar wavelets and were used in the first real-time face detector. **Paul Viola** and **Michael Jones** in their paper titled "Rapid Object Detection using a Boosted Cascade of Simple Features" used the idea of Haar-feature classifier based on the Haar wavelets. This classifier is widely used for tasks like face detection in computer vision industry.

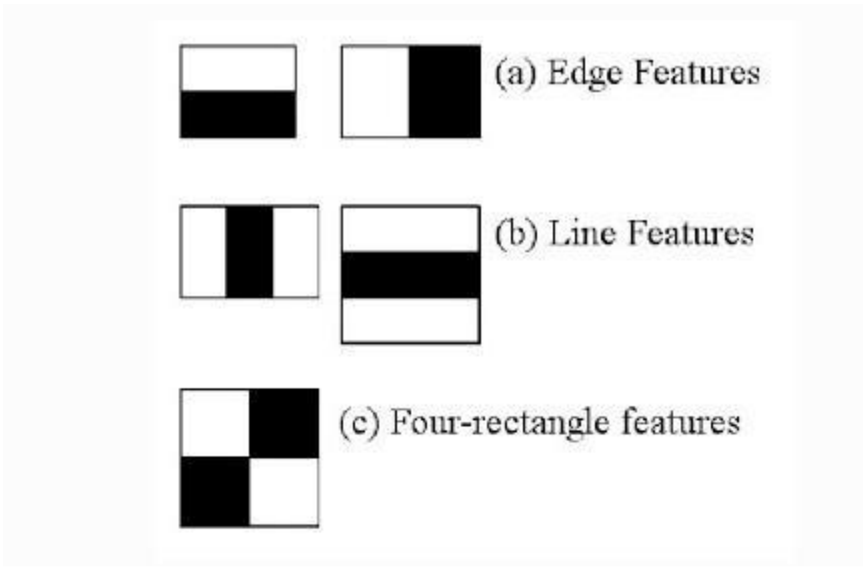
Haar cascade classifier employs a machine learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates. This can be attributed to three main reasons:

- Haar classifier employs ***'Integral Image'*** concept which allows the features used by the detector to be computed very quickly.
- The learning algorithm is based on ***AdaBoost***. It selects a small number of important features from a large set and gives highly efficient classifiers.
- More complex classifiers are combined to form a '***cascade***' which discard any non-face regions in an image, thereby spending more computation on promising object-like regions.

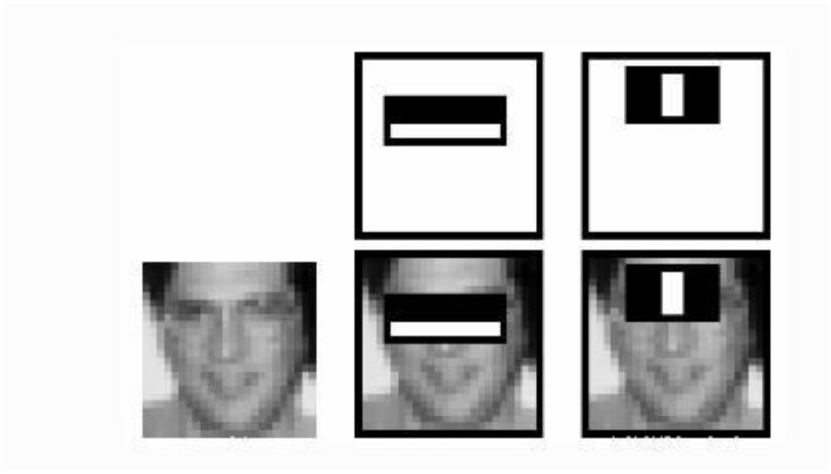
Let us now try and understand how the algorithm works on images in steps:

1. 'Haar features' extraction

After the tremendous amount of training data (in the form of images) is fed into the system, the classifier begins by extracting Haar features from each image. Haar Features are kind of convolution kernels which primarily detect whether a suitable feature is present on an image or not. Some examples of Haar features are mentioned below:



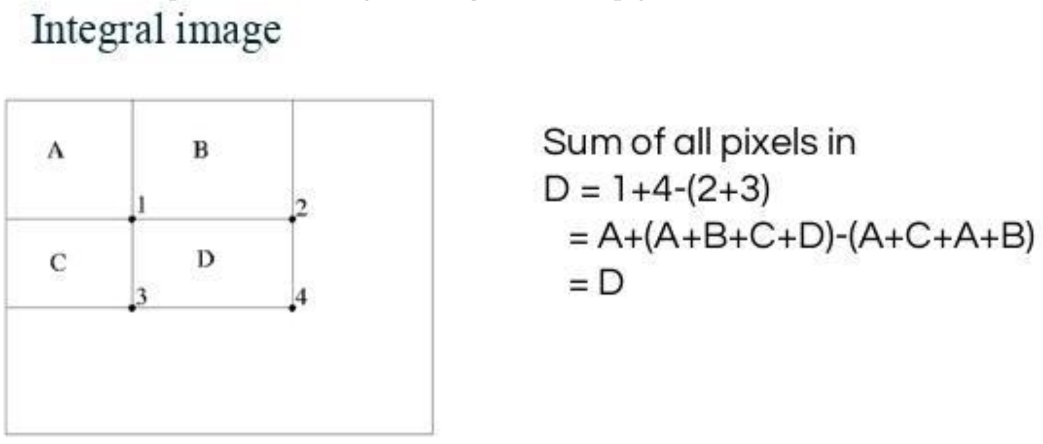
These Haar Features are like windows and are placed upon images to compute a single feature. The feature is essentially a single value obtained by subtracting the sum of the pixels under the white region and that under the black. The process can be easily visualized in the example below.



For demonstration purpose, let's say we are only extracting two features, hence we have only two windows here. The first feature relies on the point that the eye region is darker than the adjacent cheeks and nose region. The second feature focuses on the fact that eyes are kind of darker as compared to the bridge of the nose. Thus, when the feature window moves over the eyes, it will calculate a single value. This value will then be compared to some threshold and if it passes that it will conclude that there is an edge here or some positive feature.

2. 'Integral Images' concept

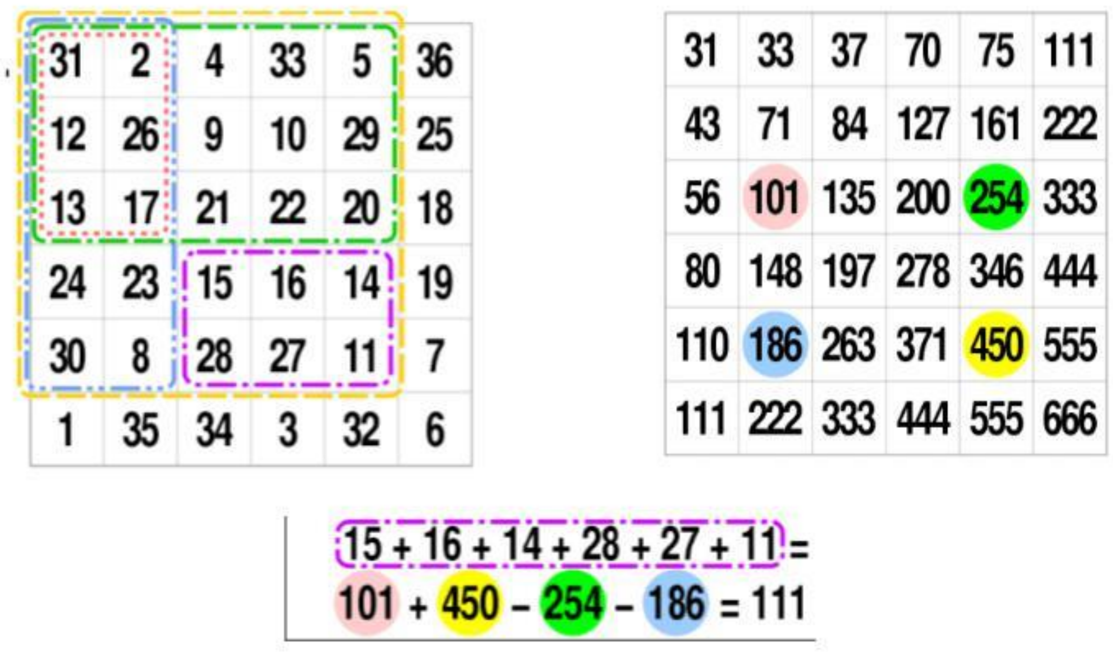
The algorithm proposed by Viola Jones uses a 24X24 base window size, and that would result in more than 180,000 features being calculated in this window. Imagine calculating the pixel difference for all the features? The solution devised for this computationally intensive process is to go for the **Integral Image** concept. The integral image means that to find the sum of all pixels under any rectangle, we simply need the four corner values.



Sum of all pixels in
D = 1+4-(2+3)
= A+(A+B+C+D)-(A+C+A+B)
= D

source

This means, to calculate the sum of pixels in any feature window, we do not need to sum them up individually. All we need is to calculate the integral image using the 4 corner values. The example below will make the process transparent.



31	2	4	33	5	36
12	26	9	10	29	25
13	17	21	22	20	18
24	23	15	16	14	19
30	8	28	27	11	7
1	35	34	3	32	6

31	33	37	70	75	111
43	71	84	127	161	222
56	101	135	200	254	333
80	148	197	278	346	444
110	186	263	371	450	555
111	222	333	444	555	666

15 + 16 + 14 + 28 + 27 + 11 =
101 + 450 – 254 – 186 = 111

source

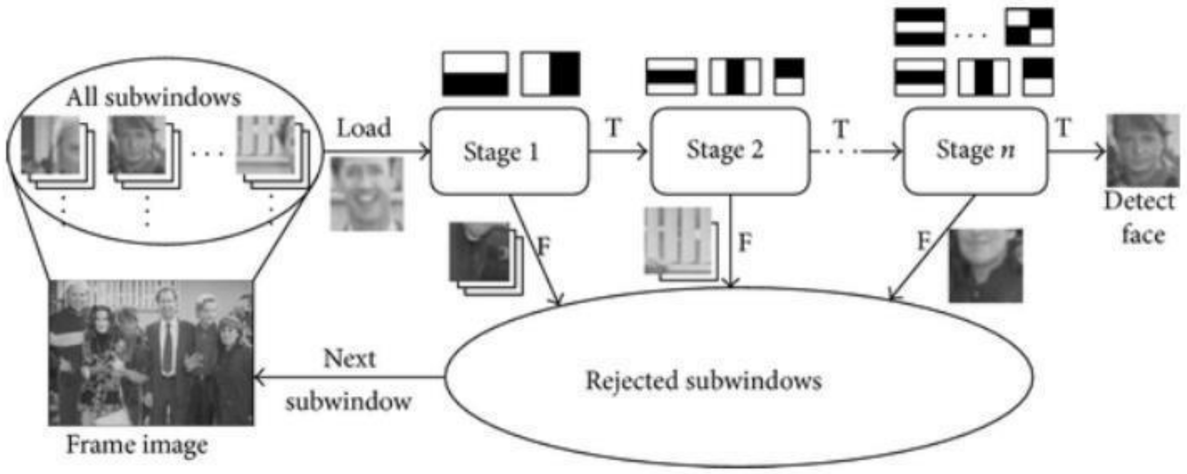
3. Pillow Module

Python Imaging Library (abbreviated as **PIL**) (in newer versions known as Pillow) is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2–2.7, with Python 3 support to be released "later".

Development appears to be discontinued with the last commit to the PIL repository coming in 2011. Consequently, a successor project called **Pillow** has forked the PIL repository and added Python 3.x support.This fork has been adopted as a replacement for the original PIL in Linux distributions including Debian and Ubuntu.

4. Using 'Cascade of Classifiers'

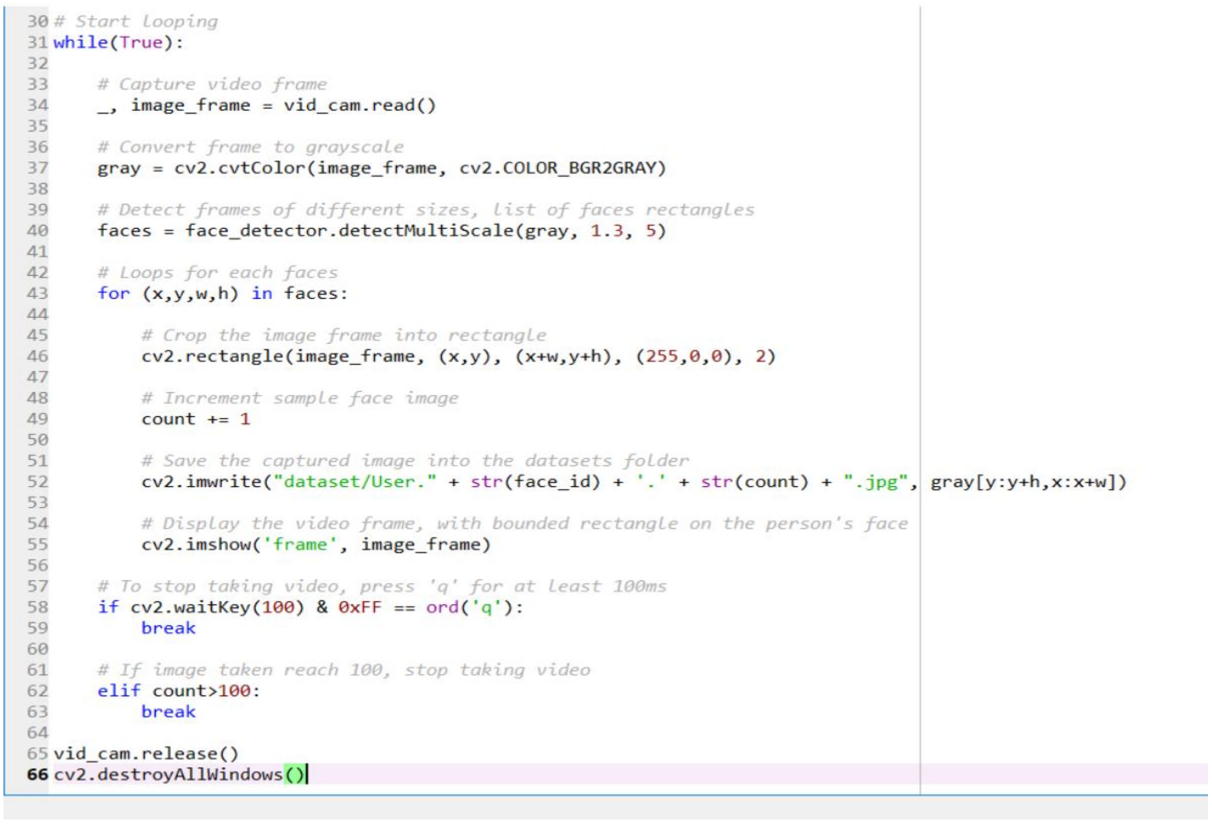
Another way by which Viola Jones ensured that the algorithm performs fast is by employing a **cascade of classifiers**. The cascade classifier essentially consists of stages where each stage consists of a strong classifier. This is beneficial since it eliminates the need to apply all features at once on a window. Rather, it groups the features into separate sub-windows and the classifier at each stage determines whether or not the sub-window is a face. In case it is not, the sub-window is discarded along with the features in that window. If the sub-window moves past the classifier, it continues to the next stage where the second stage of features is applied. The process can be understood with the help of the diagram below.



Cascade structure for Haar classifiers.

5. Data Creation and Face Detection(Source Code)

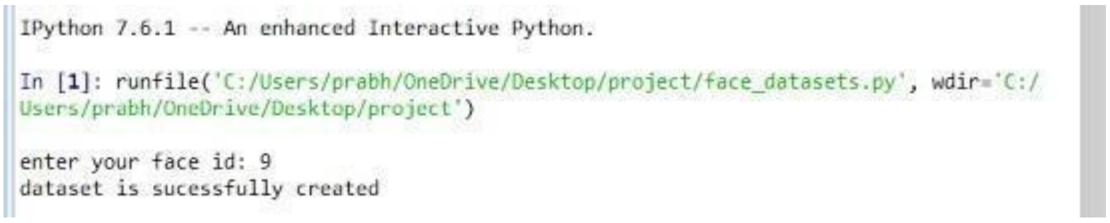
```python
import cv2
import os

def assure_path_exists(path):
    dir = os.path.dirname(path)
    if not os.path.exists(dir):
        os.makedirs(dir)

# Start capturing video
vid_cam = cv2.VideoCapture(0)

# Detect object in video stream using Haarcascade Frontal Face
face_detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# For each person, one face id
face_id = input("enter your face id")

# Initialize sample face image
count = 0

assure_path_exists("dataset/")
```



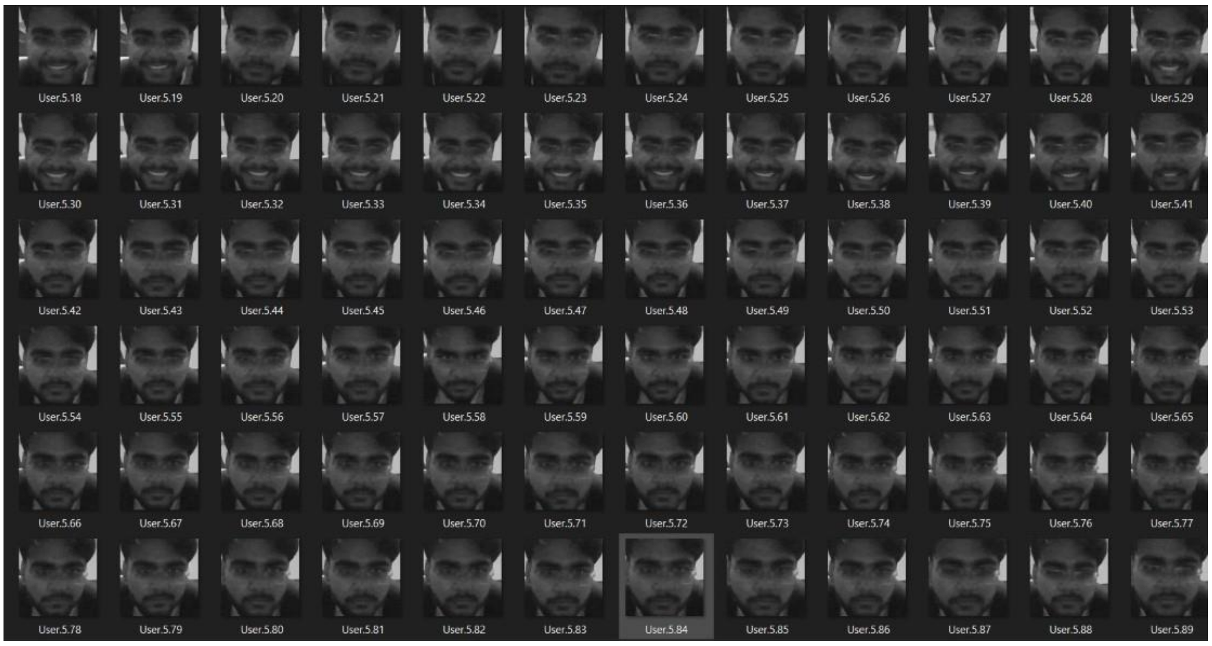
```python
# Start Looping
while(True):

    # Capture video frame
    _, image_frame = vid_cam.read()

    # Convert frame to grayscale
    gray = cv2.cvtColor(image_frame, cv2.COLOR_BGR2GRAY)

    # Detect frames of different sizes, list of faces rectangles
    faces = face_detector.detectMultiScale(gray, 1.3, 5)

    # Loops for each faces
    for (x,y,w,h) in faces:

        # Crop the image frame into rectangle
        cv2.rectangle(image_frame, (x,y), (x+w,y+h), (255,0,0), 2)

        # Increment sample face image
        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])

        # Display the video frame, with bounded rectangle on the person's face
        cv2.imshow('frame', image_frame)

    # To stop taking video, press 'q' for at least 100ms
    if cv2.waitKey(100) & 0xFF == ord('q'):
        break

    # If image taken reach 100, stop taking video
    elif count>100:
        break

vid_cam.release()
cv2.destroyAllWindows()
```

Output:

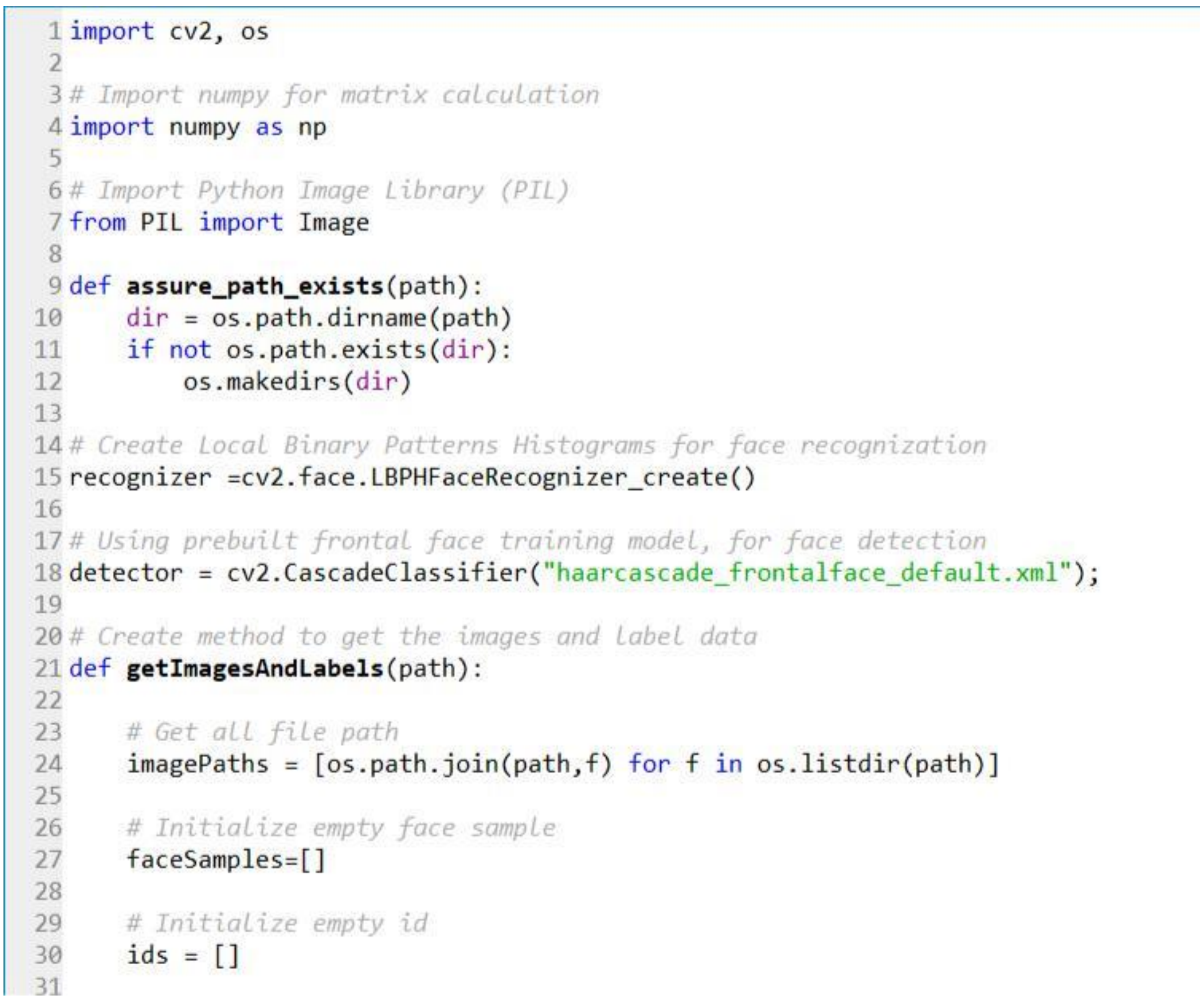
Hundred images of a person are saved in the dataset Folder.



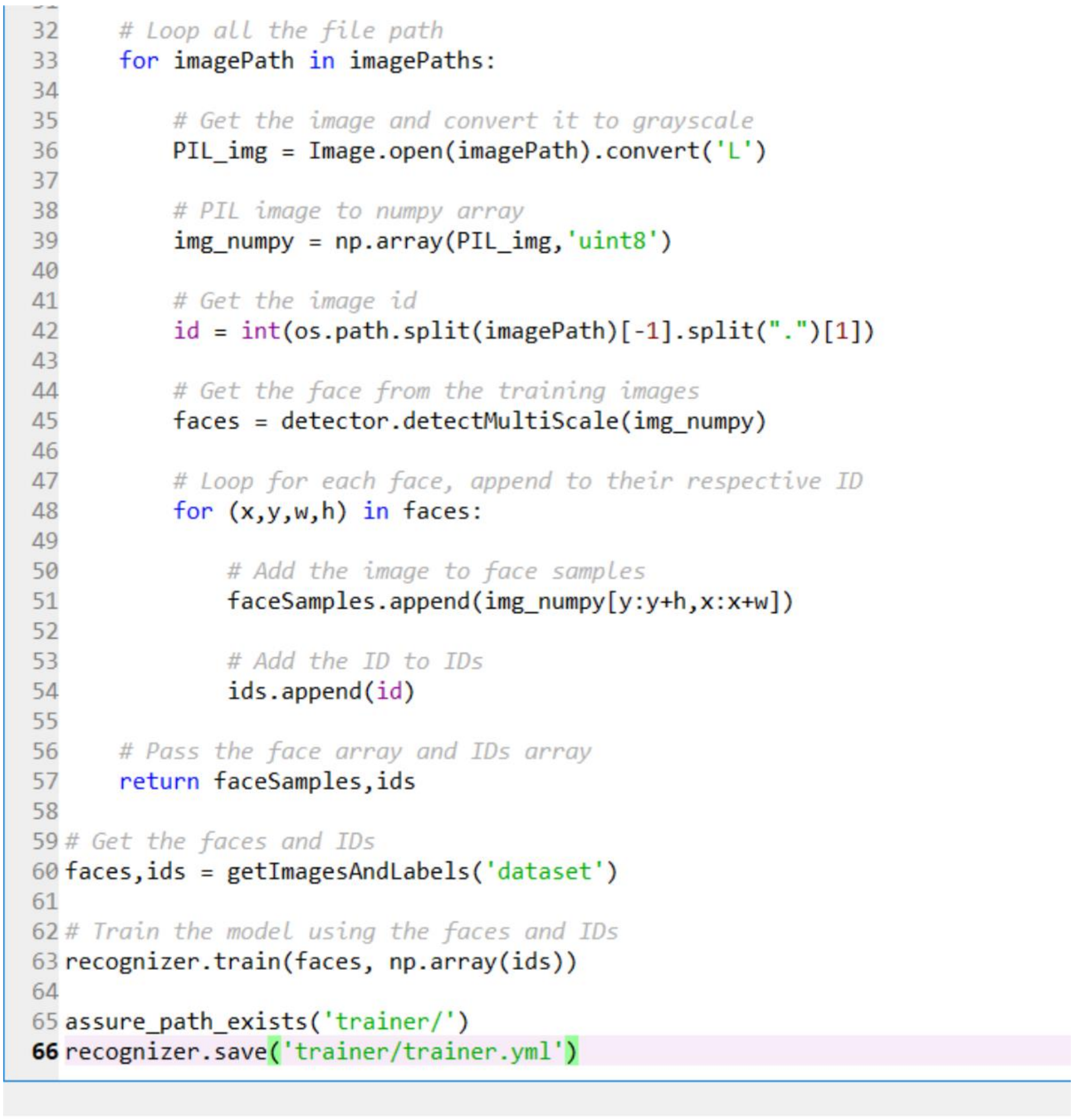
```
IPython 7.6.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/prabh/OneDrive/Desktop/project/face_datasets.py', wdir='C:/
Users/prabh/OneDrive/Desktop/project')

enter your face id: 9
dataset is sucessfully created
```



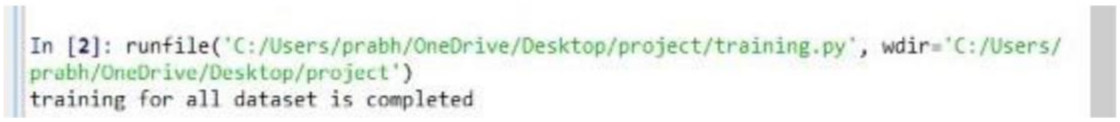
6. Training(Source Code)



```python
import cv2, os

# Import numpy for matrix calculation
import numpy as np

# Import Python Image Library (PIL)
from PIL import Image

def assure_path_exists(path):
    dir = os.path.dirname(path)
    if not os.path.exists(dir):
        os.makedirs(dir)

# Create Local Binary Patterns Histograms for face recognization
recognizer =cv2.face.LBPHFaceRecognizer_create()

# Using prebuilt frontal face training model, for face detection
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml");

# Create method to get the images and label data
def getImagesAndLabels(path):

    # Get all file path
    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]

    # Initialize empty face sample
    faceSamples=[]

    # Initialize empty id
    ids = []

```



```python
    # Loop all the file path
    for imagePath in imagePaths:

        # Get the image and convert it to grayscale
        PIL_img = Image.open(imagePath).convert('L')

        # PIL image to numpy array
        img_numpy = np.array(PIL_img,'uint8')

        # Get the image id
        id = int(os.path.split(imagePath)[-1].split(".")[1])

        # Get the face from the training images
        faces = detector.detectMultiScale(img_numpy)

        # Loop for each face, append to their respective ID
        for (x,y,w,h) in faces:

            # Add the image to face samples
            faceSamples.append(img_numpy[y:y+h,x:x+w])

            # Add the ID to IDs
            ids.append(id)

    # Pass the face array and IDs array
    return faceSamples,ids

# Get the faces and IDs
faces,ids = getImagesAndLabels('dataset')

# Train the model using the faces and IDs
recognizer.train(faces, np.array(ids))

assure_path_exists('trainer/')
recognizer.save('trainer/trainer.yml')
```



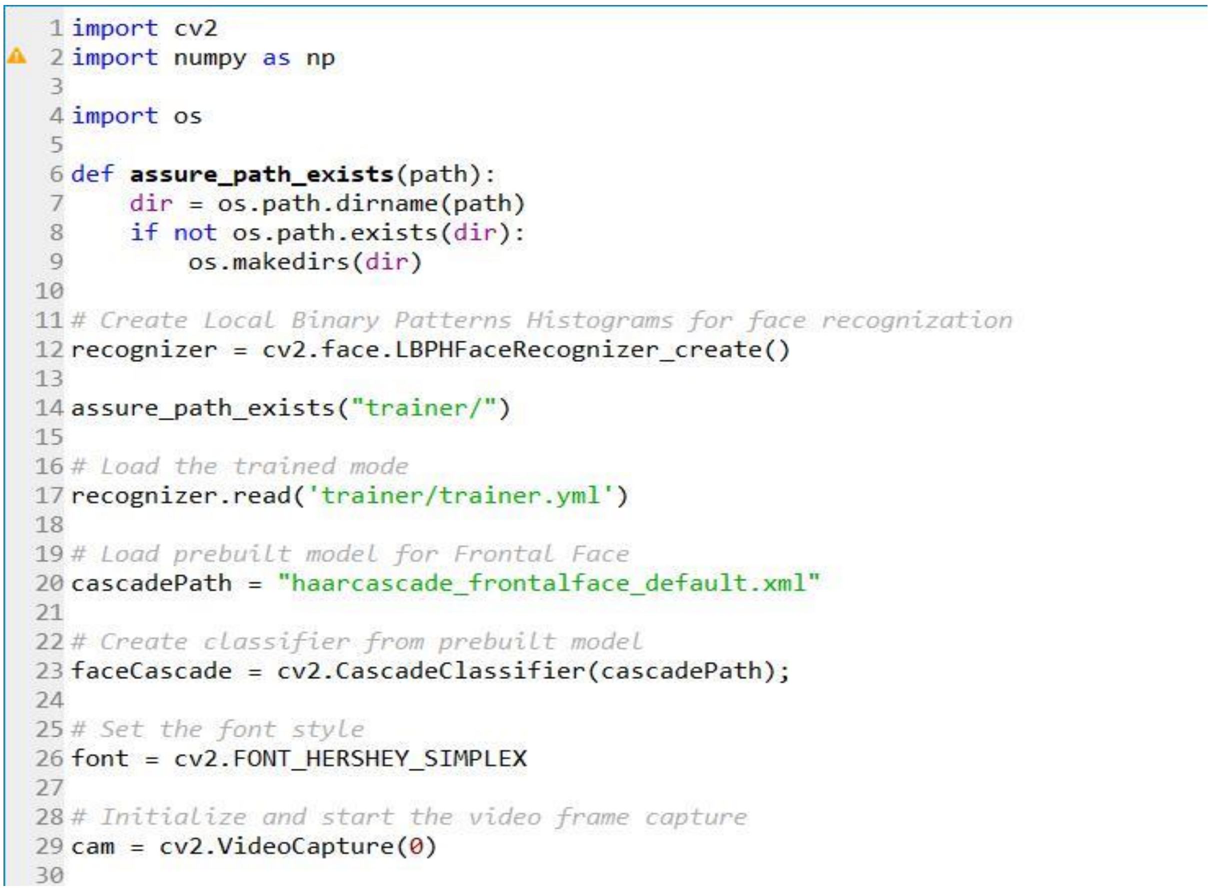
```
In [2]: runfile('C:/Users/prabh/OneDrive/Desktop/project/training.py', wdir='C:/Users/
prabh/OneDrive/Desktop/project')
training for all dataset is completed
```

8. Face Recognition

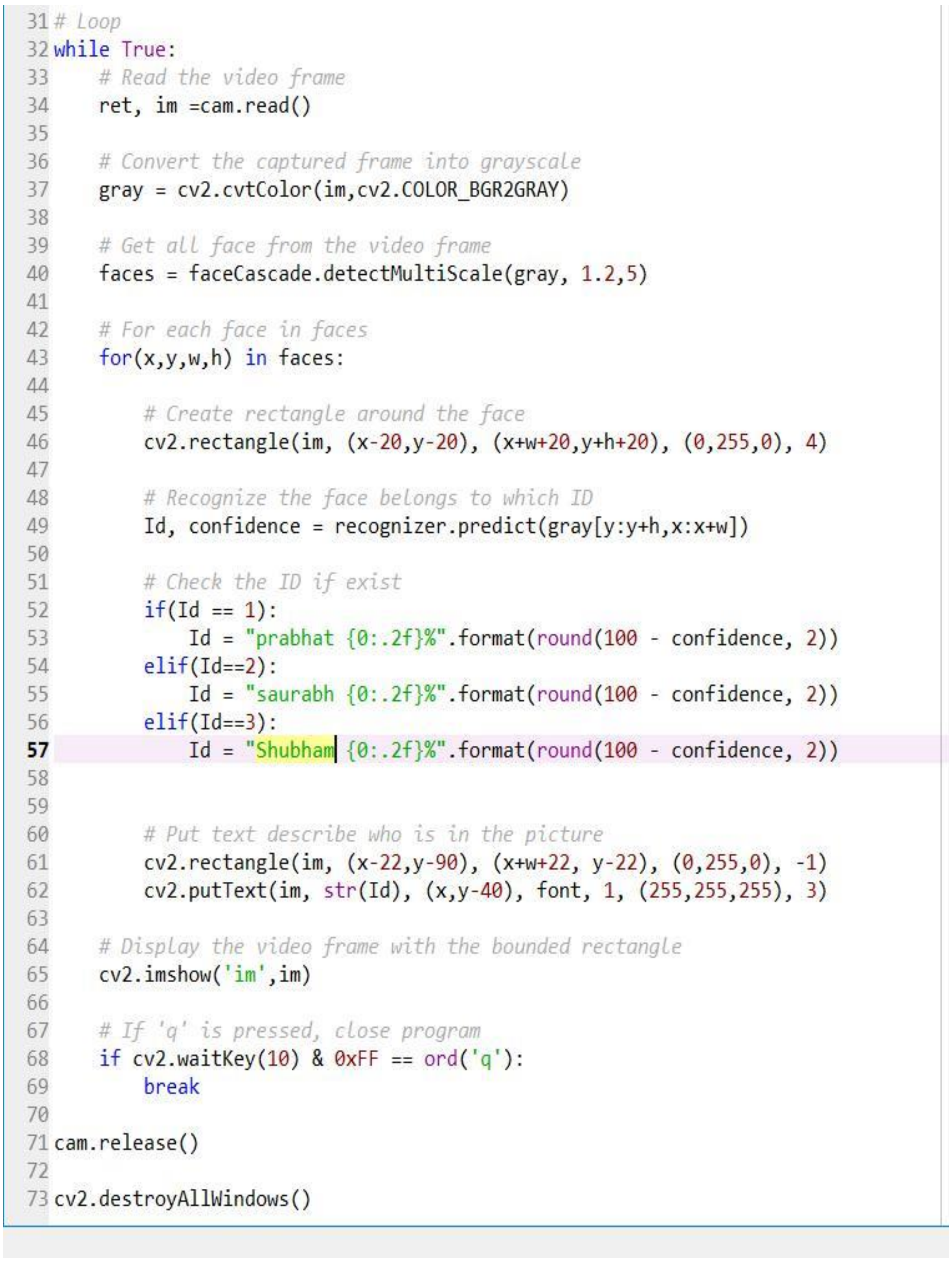
"Face recognition" redirects here. For the human cognitive process, see Face perception. For other uses, see Facial recognition.

A **facial recognition system** is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source. There are multiple methods in which facial recognition systems work, but in general, they work by comparing selected facial features from given image with faces within a database. It is also described as a Biometric Artificial Intelligence based application that can uniquely identify a person by analysing patterns based on the person's facial textures and shape.[1][*better source needed*] While initially a form of computer application, it has seen wider uses in recent times on mobile platforms and in other forms of technology, such as robotics. It is typically used as access control in security systems and can be compared to other biometrics such as fingerprint or eye iris recognition systems.[2] Although the accuracy of facial recognition system as a biometric technology is lower than iris recognition and fingerprint recognition, it is widely adopted due to its contactless and non-invasive process.[3] Recently, it has also become popular as a commercial identification and marketing tool.[4] Other applications include advanced human-computer interaction, video surveillance, automatic indexing of images, and video database, among others.

Source Code

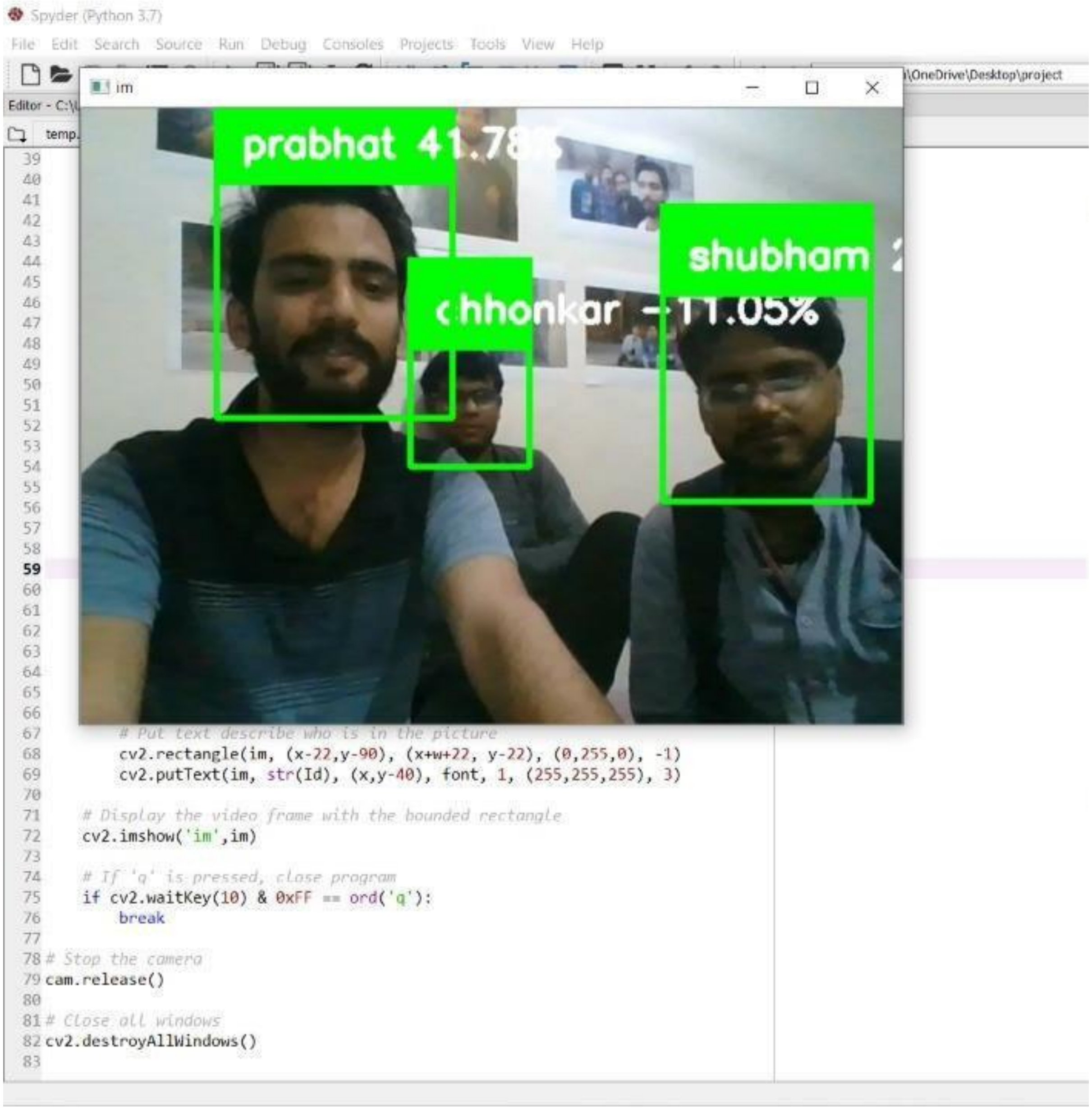


```python
import cv2
import numpy as np

import os

def assure_path_exists(path):
    dir = os.path.dirname(path)
    if not os.path.exists(dir):
        os.makedirs(dir)

# Create Local Binary Patterns Histograms for face recognization
recognizer = cv2.face.LBPHFaceRecognizer_create()

assure_path_exists("trainer/")

# Load the trained mode
recognizer.read('trainer/trainer.yml')

# Load prebuilt model for Frontal Face
cascadePath = "haarcascade_frontalface_default.xml"

# Create classifier from prebuilt model
faceCascade = cv2.CascadeClassifier(cascadePath);

# Set the font style
font = cv2.FONT_HERSHEY_SIMPLEX

# Initialize and start the video frame capture
cam = cv2.VideoCapture(0)

```



```python
# Loop
while True:
    # Read the video frame
    ret, im =cam.read()

    # Convert the captured frame into grayscale
    gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)

    # Get all face from the video frame
    faces = faceCascade.detectMultiScale(gray, 1.2,5)

    # For each face in faces
    for(x,y,w,h) in faces:

        # Create rectangle around the face
        cv2.rectangle(im, (x-20,y-20), (x+w+20,y+h+20), (0,255,0), 4)

        # Recognize the face belongs to which ID
        Id, confidence = recognizer.predict(gray[y:y+h,x:x+w])

        # Check the ID if exist
        if(Id == 1):
            Id = "prabhat {0:.2f}%".format(round(100 - confidence, 2))
        elif(Id==2):
            Id = "saurabh {0:.2f}%".format(round(100 - confidence, 2))
        elif(Id==3):
            Id = "Shubham {0:.2f}%".format(round(100 - confidence, 2))


        # Put text describe who is in the picture
        cv2.rectangle(im, (x-22,y-90), (x+w+22, y-22), (0,255,0), -1)
        cv2.putText(im, str(Id), (x,y-40), font, 1, (255,255,255), 3)

    # Display the video frame with the bounded rectangle
    cv2.imshow('im',im)

    # If 'q' is pressed, close program
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

cam.release()

cv2.destroyAllWindows()
```

Output


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
\OneDrive\Desktop\project
Editor - C:\
temp.
im
prabhat 41.78%
shubham 2
chhonkar -11.05%

```
        # Put text describe who is in the picture
        cv2.rectangle(im, (x-22,y-90), (x+w+22, y-22), (0,255,0), -1)
        cv2.putText(im, str(Id), (x,y-40), font, 1, (255,255,255), 3)

    # Display the video frame with the bounded rectangle
    cv2.imshow('im',im)

    # If 'q' is pressed, close program
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

# Stop the camera
cam.release()

# Close all windows
cv2.destroyAllWindows()
```

9. Conclusion

Face recognition technology has come a long way in the last twenty years. Today, machines are able to automatically verify identity information for secure transactions, for surveillance and security tasks, and for access control to buildings etc. These applications usually work in controlled environments and recognition algorithms can take advantage of the environmental constraints to obtain high recognition accuracy. However, next generation face recognition systems are going to have widespread application in smart environments -- where computers and machines are more like helpful assistants.

To achieve this goal computers must be able to reliably identify nearby people in a manner that fits naturally within the pattern of normal human interactions. They must not require special interactions and must conform to human intuitions about when recognition is likely. This implies that future smart environments should use the same modalities as humans, and have approximately the same limitations. These goals now appear in reach -- however, substantial research remains to be done in making person recognition technology work reliably, in widely varying conditions using information from single or multiple modalities.

10. Reference

- www.google.com
- www.wikipedia.com
- www.geeksforgeeks.com
- https://vismod.media.mit.edu/