Student Minor Research Project

REAL-TIME FACE RECOGNITION USING PYTHON AND OPENCV

Under RUSA 2.0 Scheme

(Through Ch.S.D.St.Theresa's College for Women (Autonomous), Eluru, AP)

Submitted by

Mr V R S Teja, III B.Sc. MECS (Reg.No.11705031)

Mr S Suneel Kumar, III B.Sc. MECs (Reg.No.11705052)

Under the guidance of

Mr P W Syam Babu

Assistant Professor & Project Advisor

DEPARTMENT OF COMPUTER SCIENCE

SRI Y N COLLEGE

(AUTONOMOUS)

Thrice Accredited by NAAC at 'A' Grade

Recognized by UGC as "College with Potential for Excellence"

Narsapur-534275, AP, India

December-2019

DEPARTMENT OF COMPUTER SCIENCE

SRI Y N COLLEGE

(AUTONOMOUS)

Thrice Accredited by NAAC at 'A' Grade

Recognized by UGC as "College with Potential for Excellence"

Narsapur-534275, AP, India

CERTIFICATE

This is to certify that the project work entitled **"Real-Time Face Recognition using Python and OpenCV"** *is bonafide work carried out by Mr V R S Teja (Reg.No: 11705031), Mr S Suneel Kumar (Reg.No: 11705052), submitted in Third Year of the degree B.Sc. in Computer Science during the year 2019-20 is an authentic work under my supervision and guidance.*

To the best of my knowledge, the matter embodied in the project work has not been submitted to any other College/Institute.

Date: 29-12-2019

Mr P W Syam Babu
Project Advisor
Department of Computer Science

PREFACE

Face detection is a computer vision technology that helps to locate/visualize human faces in digital images. This technique is a specific use case of object detection technology that deals with detecting instances of semantic objects of a certain class (such as humans, buildings or cars) in digital images and videos. With the advent of technology, face detection has gained a lot of importance especially in fields like photography, security, and marketing.

The main objective of this project, a real time face recognition system is capable of identifying or verifying a person from a video frame. To recognize the face in a frame, first we need to detect whether the face is present in the frame. If it is present, mark it as a Region of Interest (ROI), extract the ROI and process it for facial recognition.

ACKNOWLEDGEMENT

We place on record and warmly acknowledge the continuous encouragement, invaluable supervision, timely suggestions and inspired guidance offered by our Project advisor, ***Mr P W Syam Babu,*** *Assistant Professor, Department of Computer Science,* ***Sri Y N College (Autonomous), Narsapur*** *in bringing this report to a successful completion.*

We are grateful to ***Ms K R Sravanthi****, Assistant Professor, Department of Computer Science for permitting us to make use of the facilities available in the department to carry out the project successfully. Last but not the least we express our sincere thanks to all of our friends who have patiently extended all sorts of help for accomplishing this undertaking.*

Finally we extend our gratefulness to one and all who are directly or indirectly involved in the successful completion of this project work.

Mr V R S Teja
III.B.Sc.MECs
Reg. No 11705031

Mr S Suneel Kumar
III.B.Sc.MECs
Reg. No.11705052

DECLARATION

We, the undersigned, declare that the project entitled "**Real-Time Face Recognition using Python and OpenCV**", being submitted in Third Year of Bachelor of Science in Computer Science, Sri Y N College (Autonomous), is the work carried out by us.

Mr V R S Teja

III.B.Sc.MECs

Reg. No 11705031

Mr S Suneel Kumar

III.B.Sc.MECs

Reg. No.11705052

TABLE OF CONTENTS

Contents **Page No.**

1. SYNOPSIS

Abstract:

Face detection is a computer vision technology that helps to locate/visualize human faces in digital images. This technique is a specific use case of object detection technology that deals with detecting instances of semantic objects of a certain class (such as humans, buildings or cars) in digital images and videos. With the advent of technology, face detection has gained a lot of importance especially in fields like photography, security, and marketing.

Name of the Project: REAL-TIME FACE RECOGNITION USING PYTHON AND OPENCV

Existing System:

Face recognition biometrics is the science of programming a computer to recognize a human face. When a person is enrolled in a face recognition system, a video camera takes a series of snapshots of the face and then represents it by a unique holistic code.

• When someone has their face verified by the computer, it captures their current appearance and compares it with the facial codes already stored in the system.

• The faces match, the person receives authorization; otherwise, the person will not be identified. The existing face recognition system identifies only static face images that almost exactly match with one of the images stored in the database.

• When the current image captured almost exactly matches with one of the images stored then the person is identified and granted access.

• When the current image of a person is considerably different, say, in terms of facial expression from the images of that person which are already stored in the database the system does not recognize the person and hence access will be denied

Limitations of the Existing System

The existing or traditional face recognition system has some limitations which can be overcome by adopting new methods of face recognition:

• The existing system cannot tolerate variations in the new face image. It requires the new image to be almost exactly matching with one of the images in the database which will otherwise result in denial of access for the individual.

• The performance level of the existing system is not appreciable.

Proposed System:

Proposed System and Its Advantages

The proposed face recognition system overcomes certain limitations of the existing face recognition system. It is based on extracting the dominating features of a set of human faces stored in the database and performing mathematical operations on the values corresponding to them. Hence when a new image is fed into the system for recognition the main features are extracted and computed to find the distance between the input image and the stored images. Thus, some variations in the new face image to be recognized can be tolerated. When the new image of a person differs from the images of that person stored in the database, the system will be able to recognize the new face and identify who the person is.

The proposed system is better mainly due to the use of facial features rather than the entire face. Its advantages are in terms of:

• Recognition accuracy and better discriminatory power Computational cost because smaller images (main features) require less processing to train the LBPH.

• Because of the use of dominant features and hence can be used as an effective means of authentication

Objectives:

The main objective of this project, a real time face recognition system is capable of identifying or verifying a person from a video frame. To recognize the face in a frame, first we need to detect whether the face is present in the frame. If it is present, mark it as a Region of Interest (ROI), extract the ROI and process it for facial recognition.

Platform:

Operating System: Windows

Software Requirements:

Hands-on knowledge of Numpy and Matplotlib is essential before working on the concepts of OpenCV. Make sure that you have the following packages installed and running before installing OpenCV.

- Python
- Numpy
- Matplotlib

Hardware Requirements:

- Intel Pentium IV processor or equivalent or higher
- 2 GB RAM or Higher
- 250 GB HDD or Higher

2. SOFTWARE REQUIRMENT SPECIFICATION

Real time face recognition software

This project is divided into two parts: creating a database, and training and testing.

Creating a database

Take pictures of the person for face recognition after running create_database.py script. It automatically creates Train folder in Database folder containing the face to be recognised. You can change the name from Train to the person's name.

While creating the database, the face images must have different expressions, which is why a 0.38-second delay is given in the code for creating the data set. In this example, we take about 45 pictures/images and extract the face, convert it into grayscale and save it to the database folder with its name.

Training and testing

Training and face recognition is done next. Face_rec.py code does everything. The algorithm used here is Local Binary Patterns Histograms (LBPH).

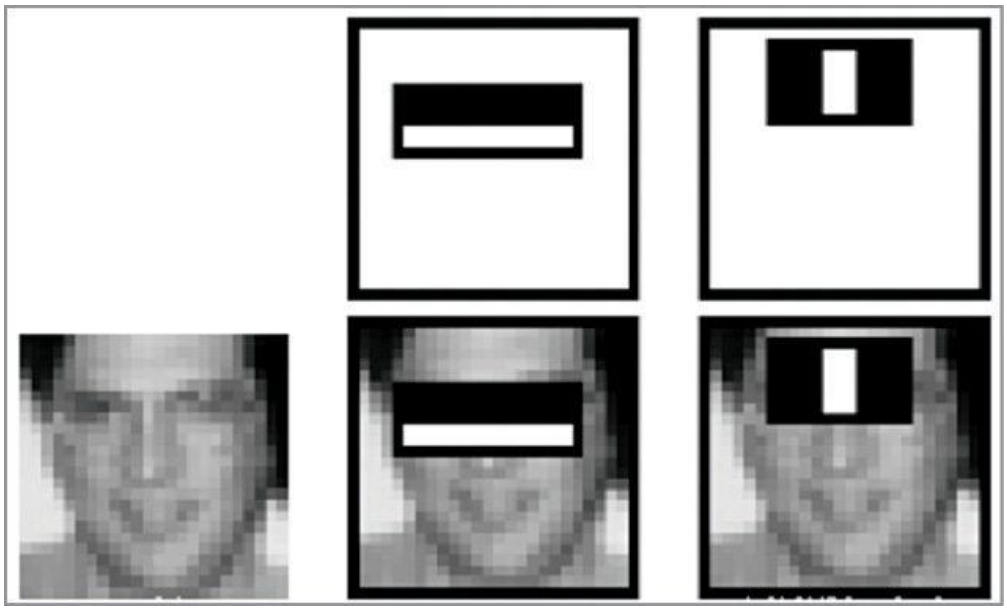


Fig. 2(a): Screenshot of Haar features

Face detection is the process of finding or locating one or more human faces in a frame or image. Haar-like feature algorithm by Viola and Jones is used for face detection. In Haar features, all human faces share some common properties. These regularities may be matched using Haar features, as shown in Fig. 1.

Two properties common to human faces are:

1. The eye region is darker than the upper cheeks.
2. The nose bridge region is brighter than the eyes.

Composition of two properties forming matchable facial features are:

1. Location and size including eyes, mouth and bridge of nose.
2. Value for oriented gradients of pixel intensities.

For example, the difference in brightness between white and black rectangles over a specific area is given by:

$$\text{Value} = \Sigma \text{ (pixels in black area)} - \Sigma \text{ (pixels in white area)}$$

The above-mentioned four features matched by Haar algorithm are compared in the image of a face shown on the left of Fig. 1.

Testing procedure

Install OpenCV and Python on Ubuntu 16.04

The project was tested on Ubuntu 16.04 using OpenCV 2.4.10. The following shell script installs all dependencies required for OpenCV and also install OpenCV 2.4.10.

$ sh ./install-opencv.sh

After installing OpenCV, check it in the terminal using import command, as shown in Fig. 2.

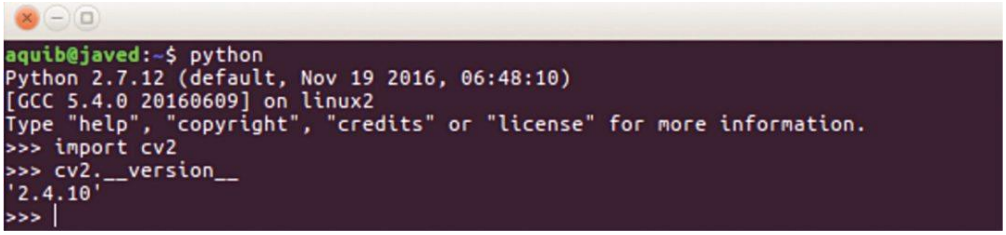


Fig. 2(b): Checking OpenCV using import command

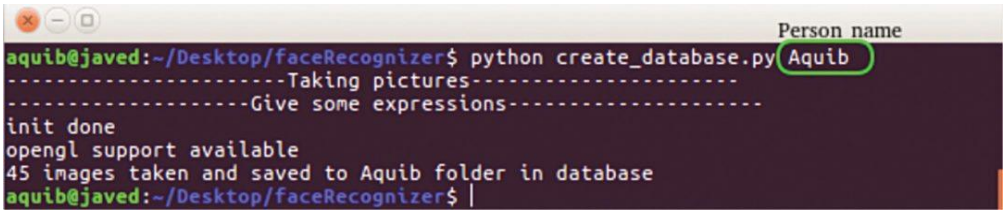


Fig. 2(c): Creating the database

1. Create the database and run the recogniser script, as given below (also shown in Fig. 3). Make at least two data sets in the database.

$ python create_database.py person_name

2. Run the recogniser script, as given below:

$ python face_rec.py

This will start the training, and the camera will open up, as shown in Fig. 4. Accuracy depends on the number of data sets as well as the quality and lighting conditions.

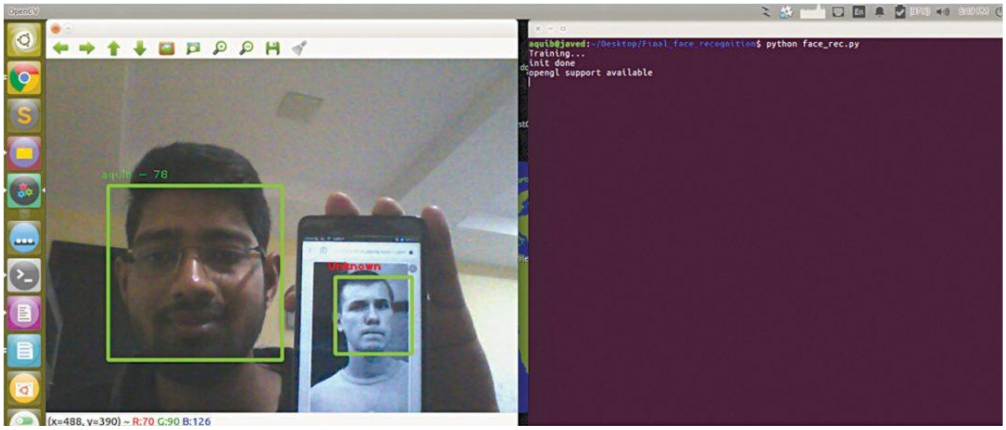


Fig. 2(d): Screenshot of face detection

Modules:

OpenCV 2.4.10.

OpenCV provides the following three face recognisers:

1. Eigenface recogniser
2. Fisherface recogniser
3. LBPH face recogniser

In this project, LBPH face recognition is used, which is createLBPHFaceRecognizer( ) function.

LBP works on gray-scale images. For every pixel in a gray-scale image, a neighbourhood is selected around the current pixel and LBP value is calculated for the pixel using the neighbourhood.

After calculating LBP value of the current pixel, the corresponding pixel location is updated in the LBP mask (it is of same height and width as input image.) with LBP value calculated, as shown in Fig. 5.

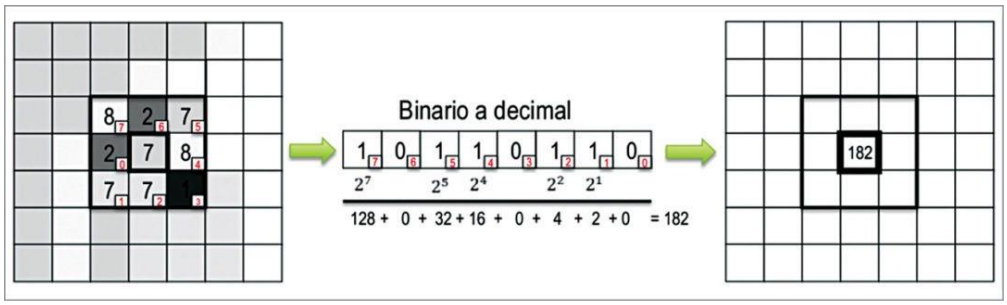


Fig. 5: Screenshot of a LBPH face recogniser

In the image, there are eight neighbouring pixels. If the current pixel value is greater than or equal to the neighbouring pixel value, the corresponding bit in the binary array is set to 1. But if the current pixel value is less than the neighbouring pixel value, the corresponding bit in the binary array is set to 0.

System and Software Design:

The final system will provide the user new tools for detecting and recognizing faces in real time video stream also provide real time video for adverse video conditions. The user will have the ability to rotate camera position, providing a wider range of view. All this features have the purpose to update older video surveillance systems. To accomplish this mission we consulted an expert

Ameri oo Santos (Surveillance Security Expert) the following requirement was the result of our interview/research.

User Requirements:

1. **Functional Requirements:**

   - The system must be able to identify human faces in live video.
   - The system must be able to search for faces in images and search for a matching face in folder, and then show the results.
   - The result should be viewed by showing the name of the face match of the input to the most-similar face in for folder.
   - The user should be able to choose (click on) to change the camera position.
   - The system must be able to draw count ours of moving objects in a live video.
   - The system must be able to display live infrared video.

2. **Non Functional Requirements:**

   - The user should be able to adjust (click on) Infra red intensity.
   - User should be able to apply green spectrum filter (click on) in a live video.
   - Critical errors and information may be shown in a textbox where it has auto-scroll when the space of the box is not enough.
   - The face should be localized by detecting inner and outer boundaries, background must be ignored.
   - The user should be able to save detected faces and names for future comparison.

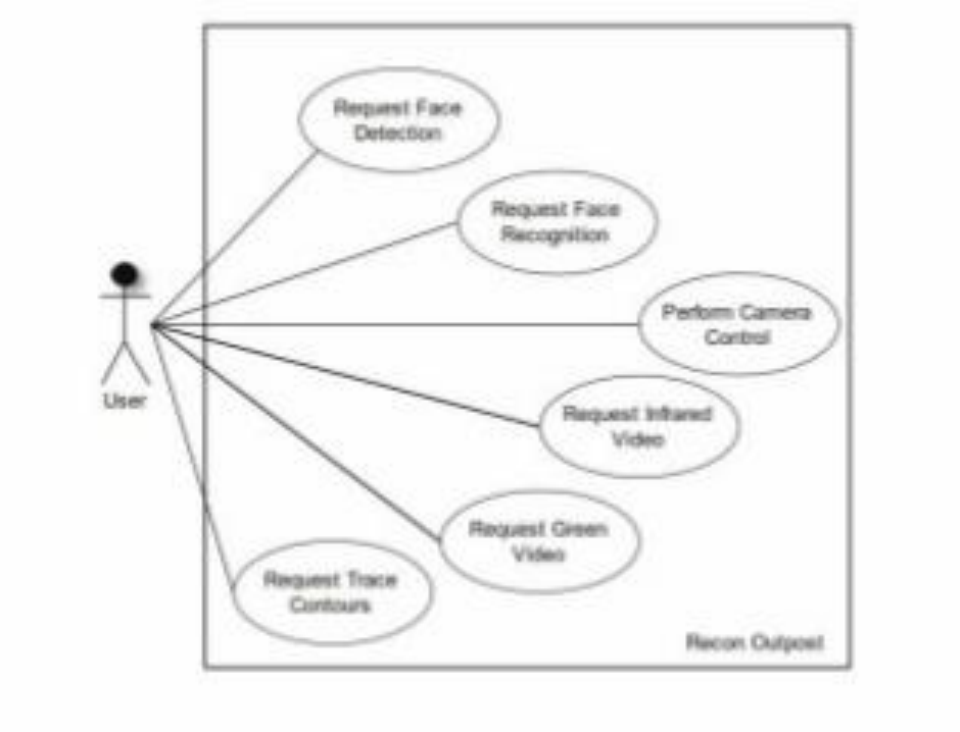


Fig 2(e): Usecase Daigram

3. TECHNOLOGY OVERVIEW

Python:

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.
- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.
- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.
- Python was designed for readability, and has some similarities to the English language with influence from mathematics.

- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

OpenCV:

OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. The document describes the so-called OpenCV 2.x API, which is essentially a C++ API, as opposed to the C-based OpenCV 1.x API (C API is deprecated and not tested with "C" compiler since OpenCV 2.4 releases)

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- **Core functionality** (**core**) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- **Image Processing** (**imgproc**) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **Video Analysis** (**video**) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- **Camera Calibration and 3D Reconstruction** (**calib3d**) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **2D Features Framework** (**features2d**) - salient feature detectors, descriptors, and descriptor matchers.
- **Object Detection** (**objdetect**) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- **High-level GUI** (**highgui**) - an easy-to-use interface to simple UI capabilities.
- **Video I/O** (**videoio**) - an easy-to-use interface to video capturing and video codecs.
- Some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

4. PROJECT DESCRIPTION

1. Introduction

Face detection is a computer vision technology that helps to locate/visualize human faces in digital images. This technique is a specific use case of object detection technology that deals with detecting instances of semantic objects of a certain class (such as humans, buildings or cars) in digital images and videos. With the advent of technology, face detection has gained a lot of importance especially in fields like photography, security, and marketing.

The directory structure of the project is as follows:

1. OpenCV-Python
   - 1.Overview
   - 2.Installation
2. Images as Arrays
   - Binary Image
   - Grayscale Image
   - Colored Image
3. Images and OpenCV
   - Importing Images in OpenCV
   - Savings images
   - Basic Operations on Images
4. Face Detection
   - Overview
   - Haar feature-based cascade classifiers
   - Face Detection with OpenCV-Python
5. Conclusion

1. OpenCV-Python

Overview

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:

1. From pre-built binaries and source :

2. 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 (check extra modules listing from OpenCV documentation)

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

2. 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.

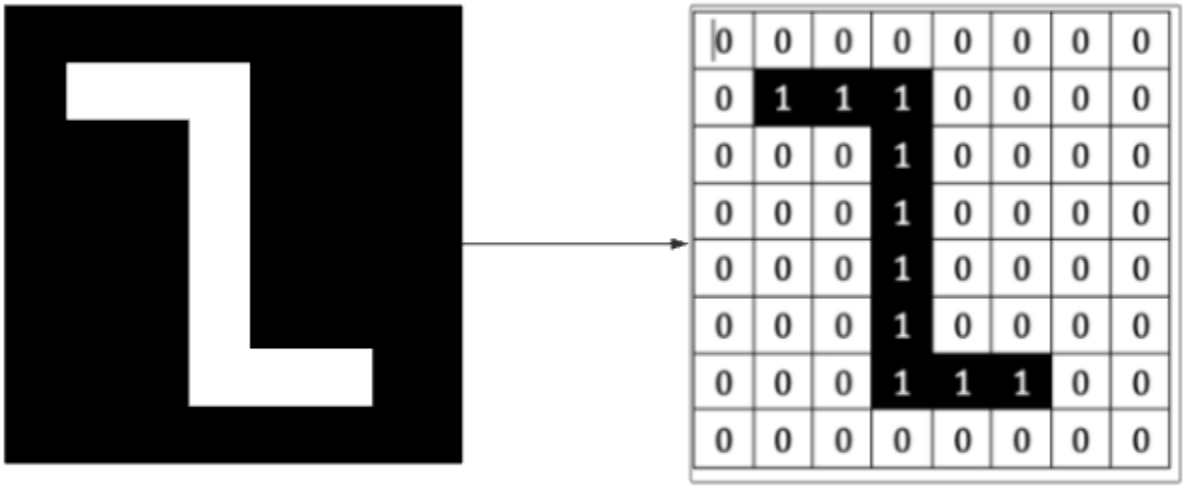


Fig 4(a): Representation of a black and white image

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.

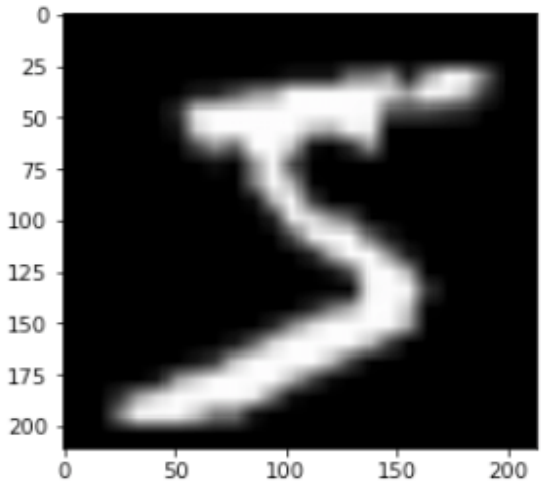


Fig 4(b): Representation of Grayscale image

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.

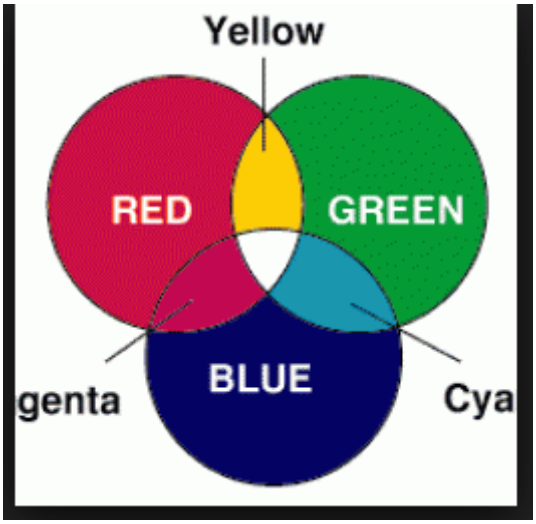


Fig 4(c): Representation of Colored image

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.

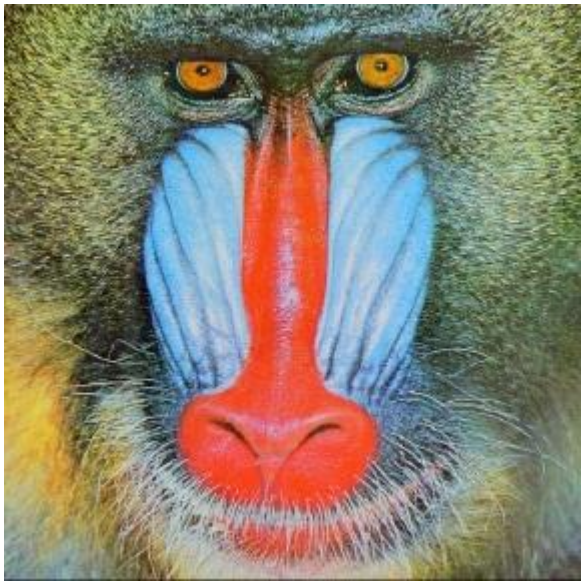


Fig 4(d): Representation of Colored image

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.

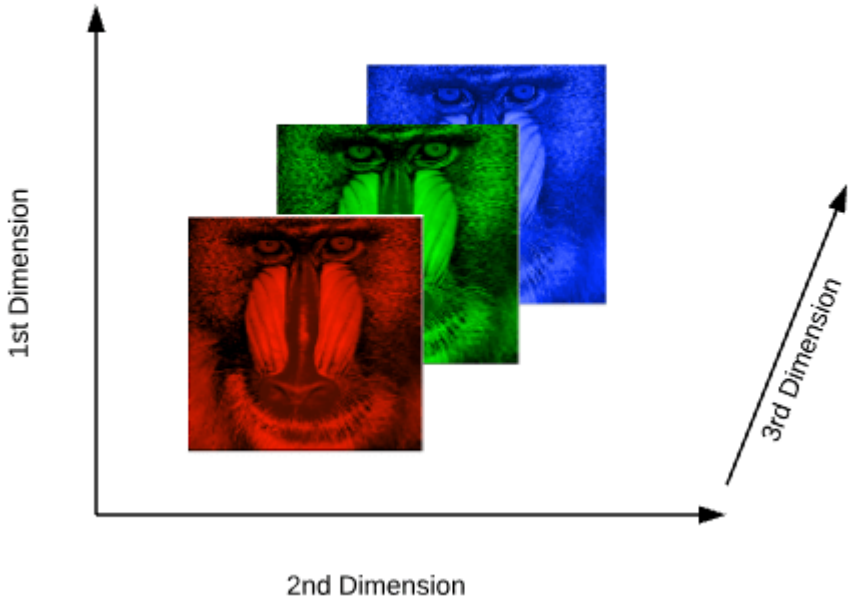


Fig 4(e): Three dimensional representation of an image

3. 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 callbacks. 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.
- img_raw = cv2.imread('image.jpg')

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

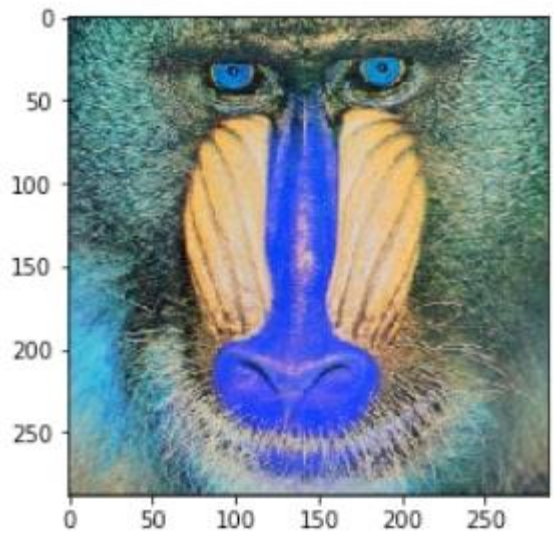
  ```
  type(img_raw)
  numpy.ndarray
  ```

  ```
  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.

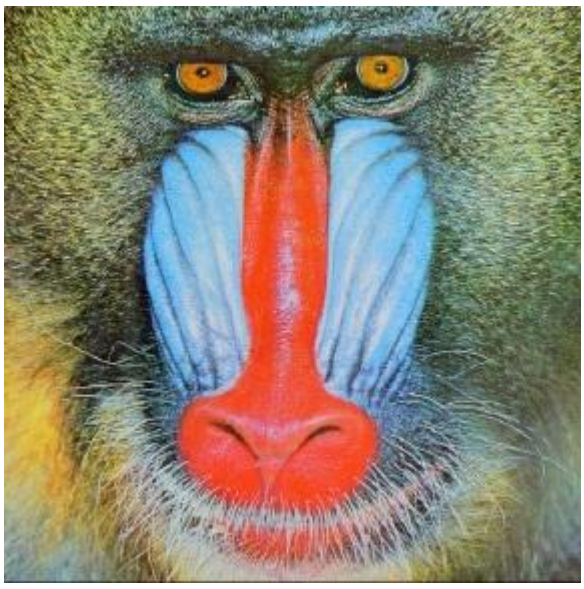
- **viewing the image**

  ```
  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.

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.

Basic Operations on Images

In this section, we will learn how we can draw various shapes on an existing image to get a flavor 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
```

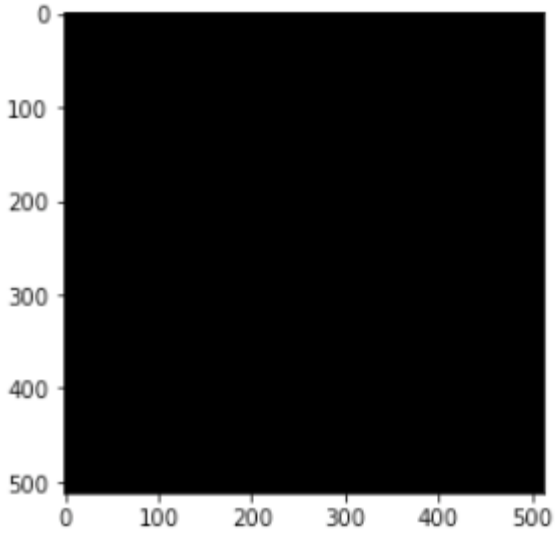
- Create a black image which will act as a template.

```
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)

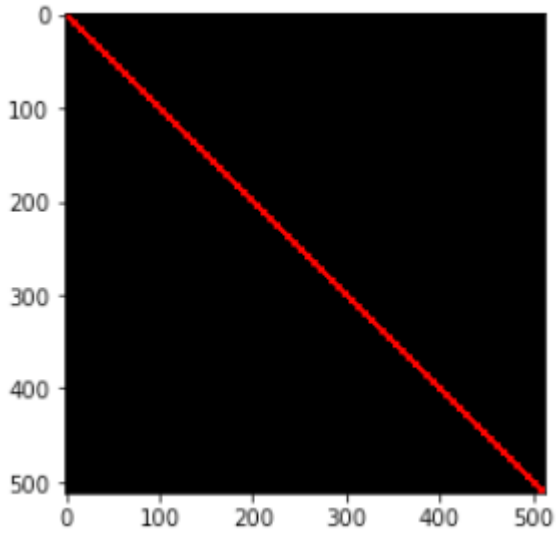
There are some common arguments which are passed in function to draw shapes on images:

- Image on which shapes are to be drawn
- co-ordinates of the shape to be drawn from Pt1(top left) to Pt2(bottom right)
- **Color**: The color 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.

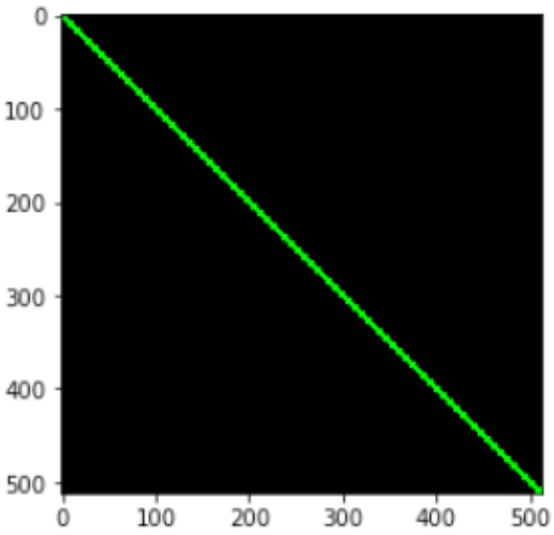
1. Straight Line

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)
```

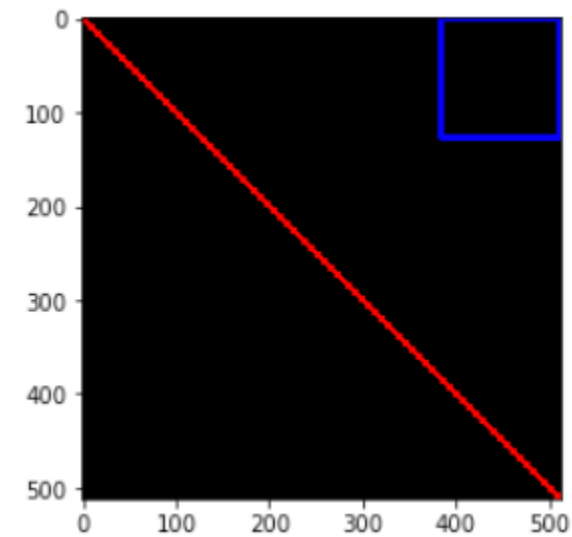


2. Rectangle

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)
```

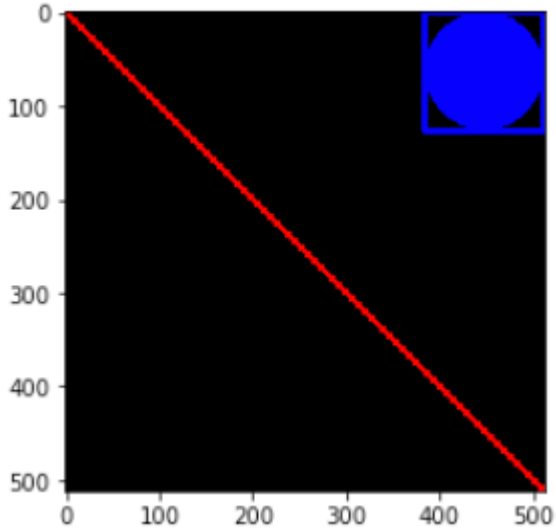


3. Circle

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)
```

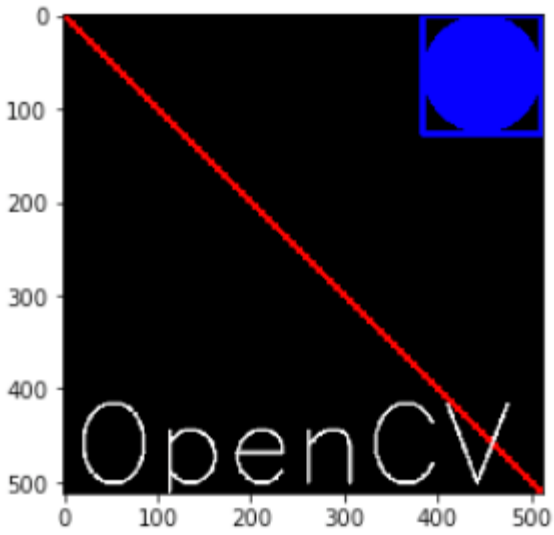


Writing on Images

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 color, 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.

4. 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.

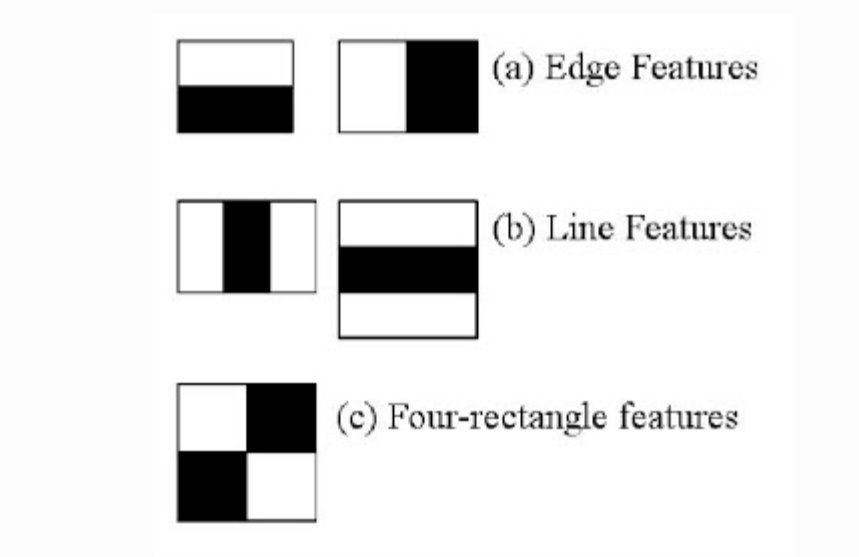
Haar cascade classifier employs a machine learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates. This can be attributed to three main reasons:

- 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.

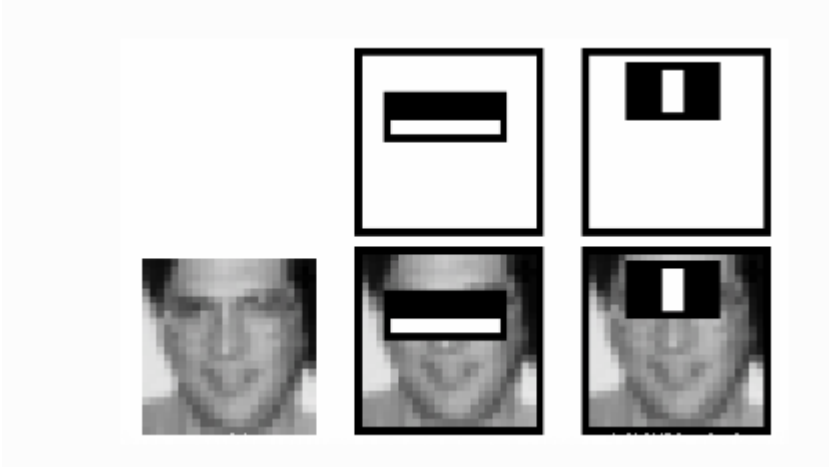
Let us now try and understand how the algorithm works on images in steps:

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.

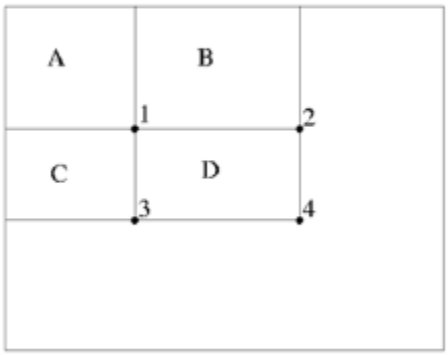


For demonstration purpose, let's say we are only extracting two features, hence we have only two windows here. The first feature relies on the point that the eye region is darker than the adjacent cheeks and nose region. The second feature focuses on the fact that eyes are kind of darker as compared 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

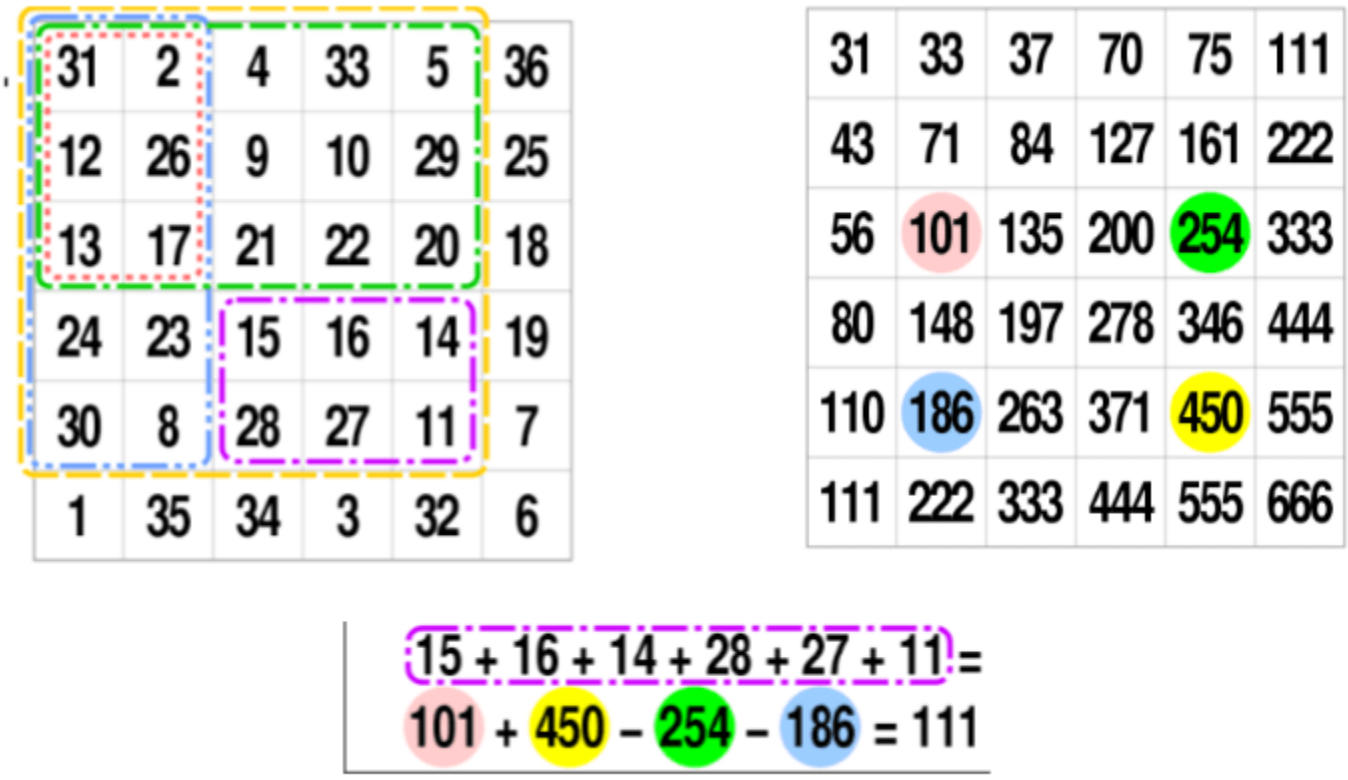
The algorithm proposed by Viola Jones uses a 24X24 base window size, and that would result in more than 180,000 features being calculated in this window. Imagine calculating the pixel difference for all the features? The solution devised for this computationally intensive process is to go for the **Integral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

Integral image



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

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.



3. 'Adaboost' : to improve classifier accuracy

As pointed out above, more than 180,000 features values result within a 24X24 window. However, not all features are useful for identifying a face. To only select the best feature out of the entire chunk, a machine learning algorithm called **Adaboost** is used. What it essentially does is that it selects only those features that help to improve the classifier accuracy. It does so by constructing a strong classifier which is a linear combination of a number of weak classifiers. This reduces the amount of features drastically to around 6000 from around 180,000.

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.

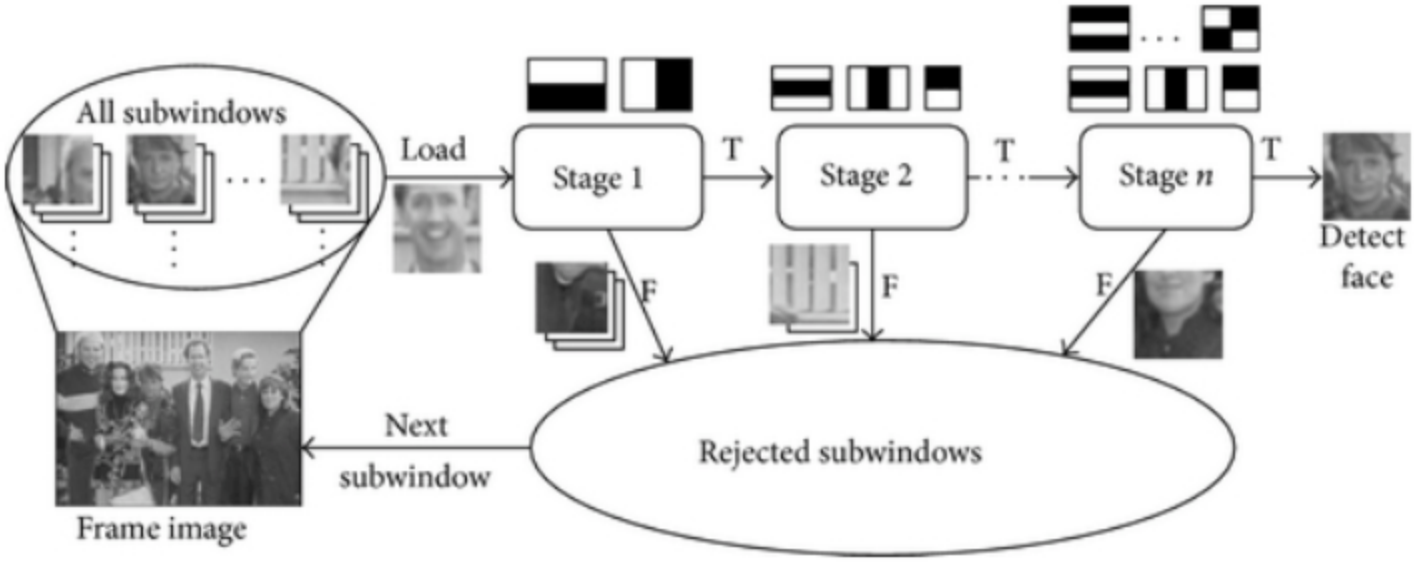


Fig: 4(f) Cascade structure for Haar classifiers.

The Paul- Viola algorithm can be visualized as follows:

Face Detection with OpenCV-Python

Now we have a fair idea about the intuition and the process behind Face recognition. Let us now use OpenCV library to detect faces in an image.

Load the necessary Libraries

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

Loading the image to be tested in grayscale

We shall be using the image below:

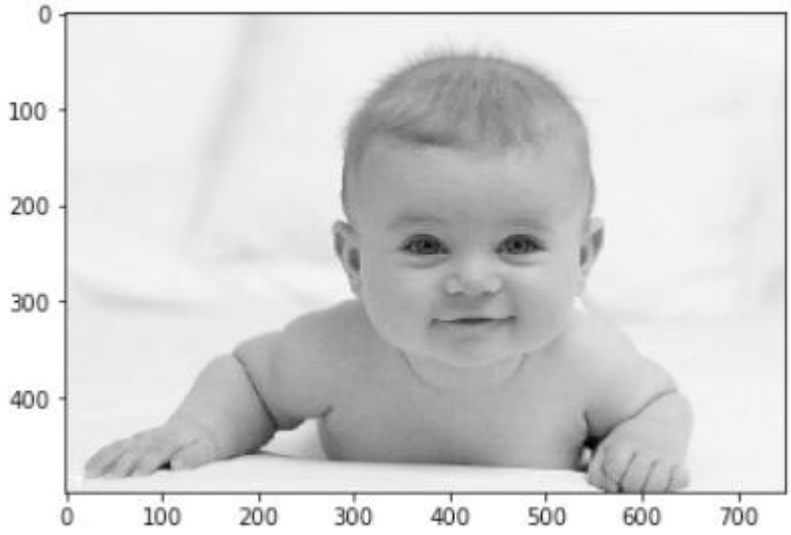
```
#Loading the image to be tested
test_image = cv2.imread('data/baby1.jpg')

#Converting to grayscale
test_image_gray = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY)

# Displaying the grayscale image
plt.imshow(test_image_gray, cmap='gray')
```

Since we know that OpenCV loads an image in BGR format, so we need to convert it into RBG format to be able to display its true colors. Let us write a small function for that.

```
<matplotlib.image.AxesImage at 0x11bb375f8>
```



Since we know that OpenCV loads an image in BGR format, so we need to convert it into RBG format to be able to display its true colors. Let us write a small function for that.

```
def convertToRGB(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

Haar cascade files

OpenCV comes with a lot of pre-trained classifiers. For instance, there are classifiers for smile, eyes, face, etc. These come in the form of xml files and are located in the opencv/data/haarcascades/ folder. However, to make things simple, you can also access them from here. Download the xml files and place them in the data folder in the same working directory as the jupyter notebook.

Loading the classifier for frontal face

```
haar_cascade_face =
cv2.CascadeClassifier('data/haarcascade/haarcascade_frontalface_default.xml')
```

Face detection

We shall be using the **detectMultiscale** module of the classifier. This function will return a rectangle with coordinates(x,y,w,h) around the detected face. This function has two important parameters which have to be tuned according to the data.

- **scalefactor** In a group photo, there may be some faces which are near the camera than others. Naturally, such faces would appear more prominent than the ones behind. This factor compensates for that.
- **minNeighbors** This parameter specifies the number of neighbors a rectangle should have to be called a face. You can read more about it here.

```
faces_rects = haar_cascade_face.detectMultiScale(test_image_gray, scaleFactor = 1.2,
minNeighbors = 5);

# Let us print the no. of faces found
print('Faces found: ', len(faces_rects))
```

```
Faces found:  1
```

Our next step is to loop over all the coordinates it returned and draw rectangles around them using Open CV. We will be drawing a green rectangle with a thickness of 2

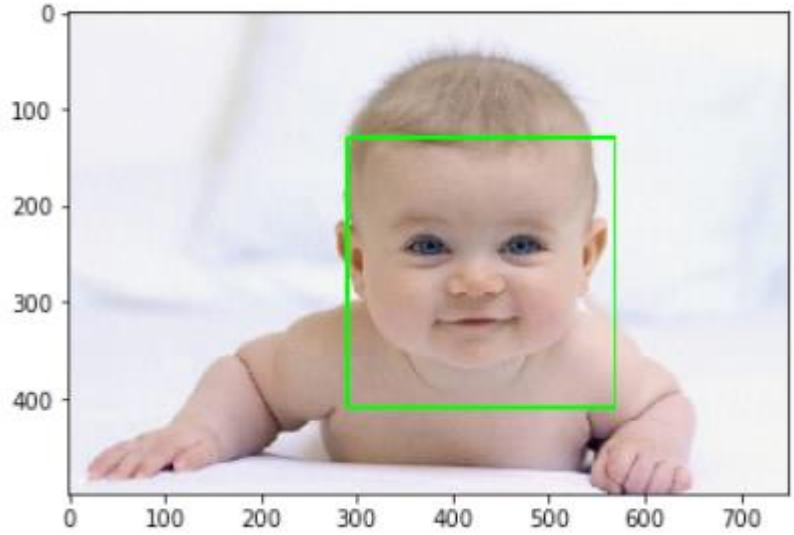
```
for (x,y,w,h) in faces_rects:
    cv2.rectangle(test_image, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

Finally, we shall display the original image in colored to see if the face has been detected correctly or not.

```
#convert image to RGB and show image

plt.imshow(convertToRGB(test_image))
```

```
<matplotlib.image.AxesImage at 0x11f22bbe0>
```



Here, it is. We have successfully detected the face of the baby in the picture. Let us now create a generalized function for the entire face detection process.

Face Detection with generalized function

```
def detect_faces(cascade, test_image, scaleFactor = 1.1):
    # create a copy of the image to prevent any changes to the original one.
    image_copy = test_image.copy()

    #convert the test image to gray scale as opencv face detector expects gray images
    gray_image = cv2.cvtColor(image_copy, cv2.COLOR_BGR2GRAY)

    # Applying the haar classifier to detect faces
    faces_rect = cascade.detectMultiScale(gray_image, scaleFactor=scaleFactor, minNeighbors=5)

    for (x, y, w, h) in faces_rect:
        cv2.rectangle(image_copy, (x, y), (x+w, y+h), (0, 255, 0), 15)

    return image_copy
```

Testing the function on new image

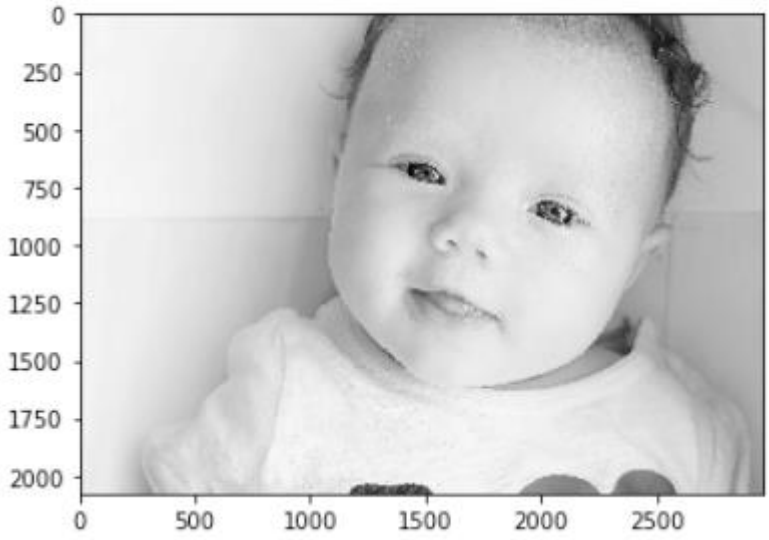
This time test image is as follows:

```
#loading image
test_image2 = cv2.imread('baby2.jpg')

# Converting to grayscale
test_image_gray = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY)

# Displaying grayscale image
plt.imshow(test_image_gray, cmap='gray')
```

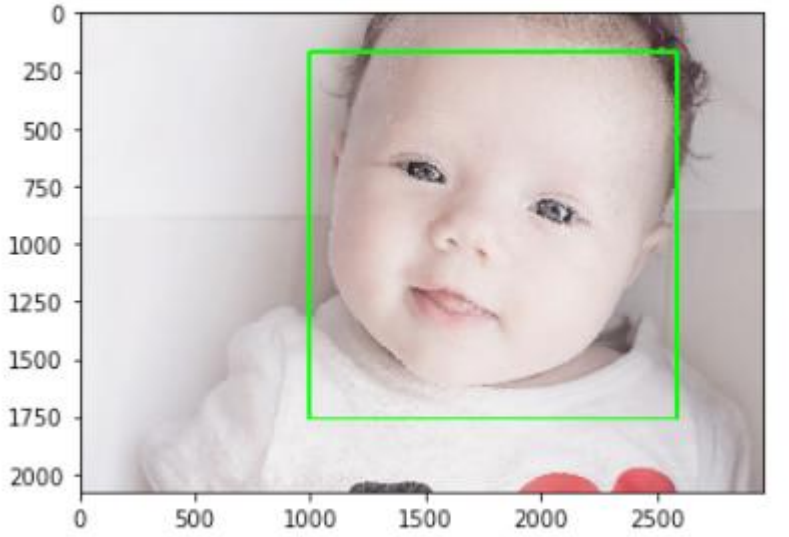
```
<matplotlib.image.AxesImage at 0x11f065be0>
```




```
#call the function to detect faces
faces = detect_faces(haar_face_cascade, test_image2)
```

```
#convert to RGB and display image
plt.imshow(convertToRGB(faces))
```

```
<matplotlib.image.AxesImage at 0x11f11a160>
```



Testing the function on a group image

Let us now see if the function works well on a group photograph or not. We shall be using the picture below for our purpose.

Image: The Indian Women's Cricket Team.

```
#loading image
test_image2 = cv2.imread('group.jpg')

#call the function to detect faces
faces = detect_faces(haar_cascade_face, test_image2)

#convert to RGB and display image
```

plt.imshow(convertToRGB(faces))

Conclusion

In this tutorial, we learned about the concept of face detection using Open CV in Python using Haar cascade. There are a number of detectors other than the face, which can be found in the library. Feel free to experiment with them and create detectors for eyes, license plates, etc. For any queries, please leave a comment below.

Source Code

```
# facerec.py
import cv2, sys, numpy, os,pyttsx,time
size = 4
fn_haar = 'haarcascade_frontalface_default.xml'
fn_dir = 'att_faces'


# Part 1: Create fisherRecognizer
print('Training...')
# Create a list of images and a list of corresponding names
(images, lables, names, id) = ([], [], {}, 0)
for (subdirs, dirs, files) in os.walk(fn_dir):
    for subdir in dirs:
        names[id] = subdir
        subjectpath = os.path.join(fn_dir, subdir)
        for filename in os.listdir(subjectpath):
            path = subjectpath + '/' + filename
            lable = id
       images.append(cv2.imread(path, 0))
            lables.append(int(lable))
        id += 1
(im_width, im_height) = (112, 92)

# Create a Numpy array from the two lists above
(images, lables) = [numpy.array(lis) for lis in [images, lables]]

# OpenCV trains a model from the images

model = cv2.face.createFisherFaceRecognizer()
model.train(images, lables
```

5. SNAPSHOTS

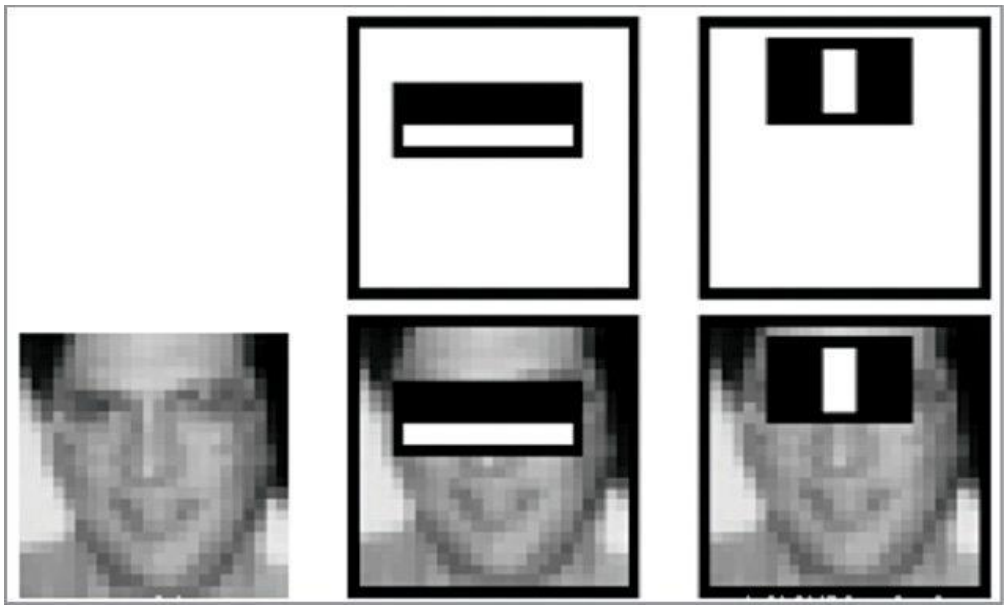


Fig. 5(a): Screenshot of Haar features

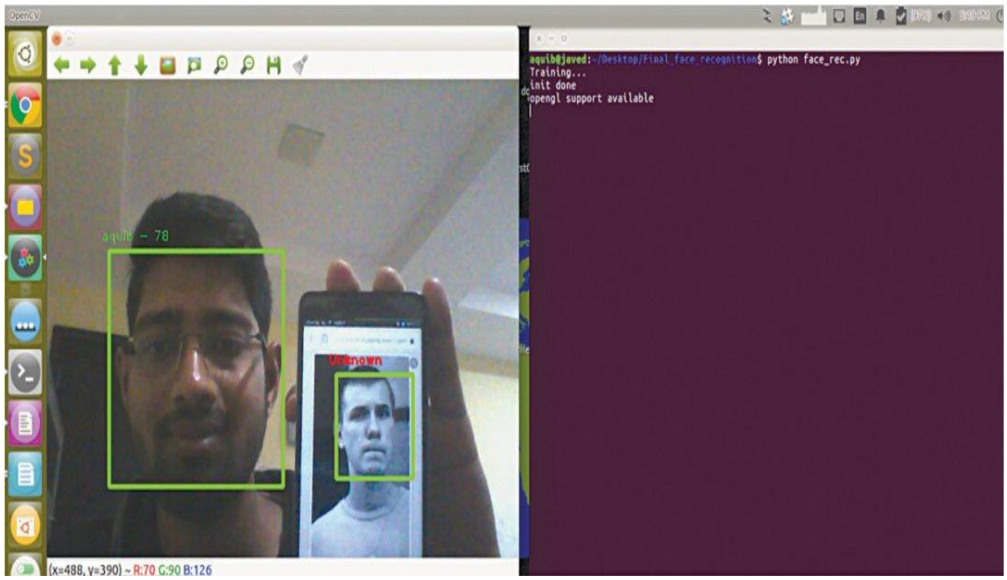


Fig.5(b) : Screenshot of Facedetection

6. SCOPE OF THE PROJECT

Scope of the Research Scope of the system is completely identification of the face. Within the allocated time completing the system with the specified user requirements. One system is for the administrator and the other one is for the users. It can be used in many fields there are Bank, Hotel and Police Station. A throughout survey has revealed that various methods and combination of these methods can be applied in development of a new face recognition system. Among the many possible approaches, we have decided to use a combination of knowledge based methods for face detection part and neural network approach for face recognition part.

The main reason in this selection is their smooth applicability and reliability issues.

7. CONTRIBUTION IN THE PROJECT

Facial recognition can aid forensic investigations by automatically recognizing individuals in security footage or other videos. Face **recognition** software can also be **used** to identify dead or unconscious individuals at crime scenes.

Facial recognition is the process of identifying or verifying the identity of a person using their **face**. It captures, analyzes, and compares patterns based on the person's **facial** details. The **face detection** process is an essential step as it detects and locates human faces in images and videos.

Our project Real-Time Face Recognition Using Python And Opencv was developed by all three of us.We, a team of three persons took a step by step approach in order to reach our goal.We applied the knowledge we gained during our training period at **Sri Y N College** and developed this project **"REAL-TIME FACE RECOGNITION USING PYTHON AND OPENCV"**.

8. CONCLUSION

Face recognition is still a challenging problem in the field of computer vision. It has received a great deal of attention over the past years because of its several applications in various domains. Although there is strong research effort in this area, face recognition systems are far from ideal to perform adequately in all situations form real world. Paper presented a brief survey of issues methods and applications in area of face recognition. There is much work to be done in order to realize methods that reflect how humans recognize faces and optimally make use of the temporal evolution of the appearance of the face for recognition

9. BIBLIOGRAPHY

- Ahonen, Timo, Abdenour Hadid, and Matti Pietikainen. "**Face description with local binary patterns: Application to face recognition**." *IEEE transactions on pattern analysis and machine intelligence* 28.12 (2006): 2037–2041.

- Ojala, Timo, Matti Pietikainen, and Topi Maenpaa. "**Multiresolution gray-scale and rotation invariant texture classification with local binary patterns**." *IEEE Transactions on pattern analysis and machine intelligence* 24.7 (2002): 971–987.

- Ahonen, Timo, Abdenour Hadid, and Matti Pietikäinen. "**Face recognition with local binary patterns**." *Computer vision-eccv 2004* (2004): 469–481.

- LBPHOpenCV: https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#local-binary-patterns-histograms

- Local Binary Patterns: http://www.scholarpedia.org/article/Local_Binary_Patterns