Face Recognition with Python using OpenCV

Presented to

Phylip rechard
Lecturer of City University
CSE Department

Presented by

Nusrat Jahan Muna
Id:161412314
Raveya Khatun
Id:161412333
Nusrat Jahan
id:161412344
MD.Al-Amin Pramanik
Id:161412344

Abstract

Face recognition has been one of the most interesting and important research fields in the past two decades. The reasons come from the need of automatic recognitions and surveillance systems, the interest in human visual system on face recognition, and the design of human-computer interface, etc. These researches involve knowledge and researchers from disciplines such as neuroscience, psychology, computer vision, pattern recognition, image processing, and machine learning, etc. A bunch of papers have been published to overcome difference factors (such as illumination, expression, scale, pose, ……) and achieve better recognition rate, while there is still no robust technique against uncontrolled practical cases which may involve kinds of factors simultaneously. In this report, we'll go through general ideas and structures of recognition, important issues and factors of human faces, critical techniques and algorithms, and finally give a comparison and conclusion. Readers who are interested in face recognition could also refer to published surveys [1-3] and website about face recognition [4]. To be announced, this report only focuses on color-image-based (2D) face recognition, rather than video-based (3D) and thermal-image-based methods.

1.0 Introduction:

Object detection [9] and location in digital images has become one of the most important applications for industries to ease user, save time and to achieve parallelism. This is not a new technique but improvement in object detection is still required in order to achieve the targeted objective more efficiently and accurately. The main aim of studying and researching computer vision is to simulate the behavior and manner of human eyes directly by using a computer and later on develop a system that reduces human efforts. Computer vision is such kind of research field which tries to perceive and represents the 3D information for world objects. Its main purpose is reconstructing the visual aspects of 3D objects after analyzing the 2D information extracted. Real life 3D objects are represented by 2D images. The process of object detection analysis is to determine the number, location, size, position of the objects in the input image. Object detection is the basic concept for tracking and recognition of objects, which affects the efficiency and accuracy of object recognition. The common object detection method is the color-based approach, detecting objects based on their color values [4]. The method is used because of its strong adaptability and robustness, however, the detection speed needs to be improved, because it requires testing all possible windows by exhaustive search and has high computational complexity. Object detection from a complex background is a challenging application in image processing. The goal of this project is to identify objects placed over a surface from a complex background image using various techniques. The detection of the objects can be extended using automation and robotics for plucking of the objects like apples, bananas from the corresponding tree using the image processing techniques and it will be easier, faster and convenient to pluck the apples and bananas rather than the manual plucking. The standard performance measure that is commonly used for the object category segmentation problem is called Intersection-over-Union (IOU) [3].

1.1 Problem Statement

Every object class has its own special features that help in classifying the object. Object recognition is that sub-domain of computer vision which helps in identifying objects in an image or video sequence. With more efficient algorithms, objects can even be recognized even when they are partially obstructed from the direct view. Various approaches to this task have been implemented in the past years. Various terms related to object detection are:

Edge matching

- Uses edge detection techniques to find the edges
- Effect of changes in lighting and color
- Count the number of overlapping edges.

Divide and Conquer search

- All positions are to be considered as a set.
- The lower bound is determined at best position in the cell.
- The cell is pruned if the bound is too large.
- The process stops when a cell becomes small enough.

Grayscale matching

- Edges give a lot of information being robust to illumination changes.
- Pixel distance is computed as a function of both pixel intensity and position. The same thing can compute with color too.

1.2 Proposal solution

1.2.1 OpenCV

(Open Source Computer Vision) is an open source computer vision and machine learning software library [10]. OpenCV was initially built to provide a common infrastructure for applications related to computer vision and to increase the use of machine perception in the commercial products. As it is a BSD-licensed product so it becomes easy for businesses to utilize and modify the existing code in OpenCV. Around 3000 algorithms are currently embedded inside OpenCV library, all these algorithms being efficiently optimized. It supports real-time vision applications. These algorithms are categorized under classic algorithms, state of art computer vision algorithms and machine learning algorithms. These algorithms are easily implemented in Java, MATLAB, Python, C, C++ etc. and are well supported by operating system like Window, Mac OS, Linux and Android. A full-featured CUDA and OpenCL interfaces are being actively developed for the betterment of technology. There are more than 500 different algorithms and even more such functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers. For OpenCV to work efficiently with python 2.7 we need to install NumPy package first

2.0 Background Study

2.1 Install OpenCV-Python

Installation

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

2.1.1 Unofficial pre-built OpenCV packages for Python.

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

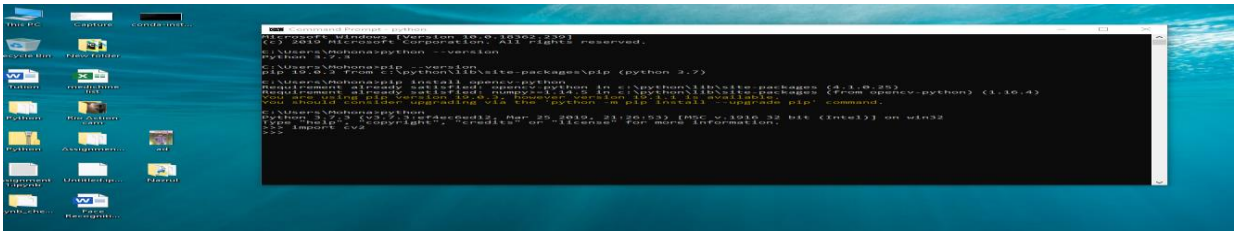
run pip install opencv-python if you need only main modules

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.

.

Download OpenCV from Sourceforge. Go to OpenCV/build/python/2.7 folder. Copy cv2.pyd to C:/Python27/lib/site- packages.



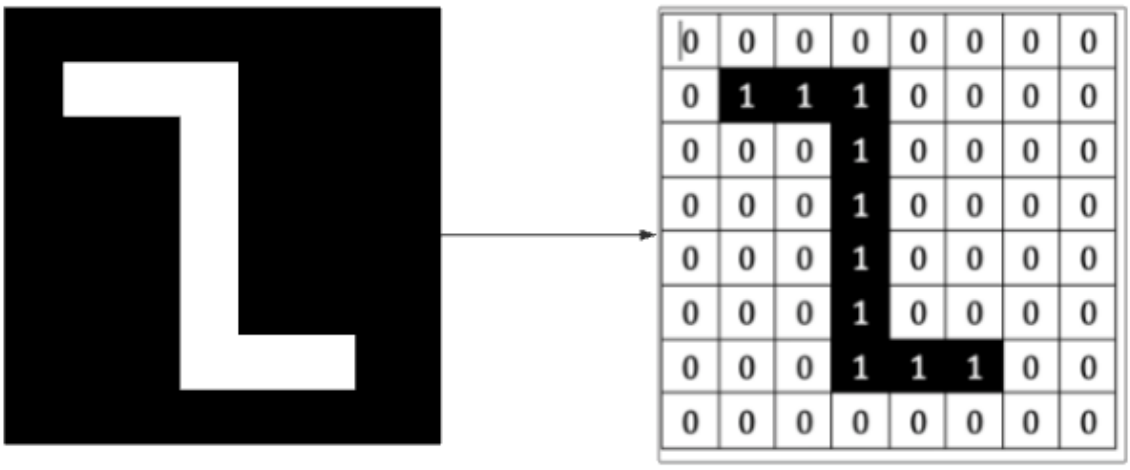
2.2 Images as Arrays

Read an Image Use the function CV2.imread() to read an image. The image should be in the current working directory otherwise, we need to specify the full path of the image as the first argument. The second argument is a flag which specifies the way image should be read.

1. CV2.IMREAD_COLOR: This function is used to load a color image. Transparency of image, if present will be neglected. It is the default flag.
2. CV2.IMREAD_GRAYSCALE: Loads image in grayscale mode
3. CV2.IMREAD_UNCHANGED

1. Binary Image

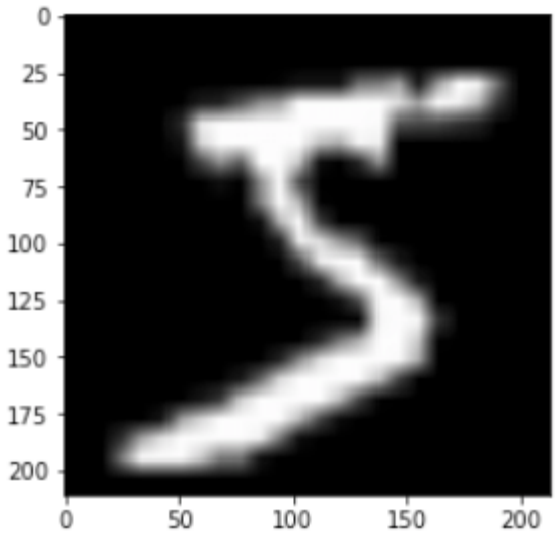
A binary image consists of 1 bit/pixel and so can have only two possible colors, i.e., black or white. Black is represented by the value 0 while 1 represents white.



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

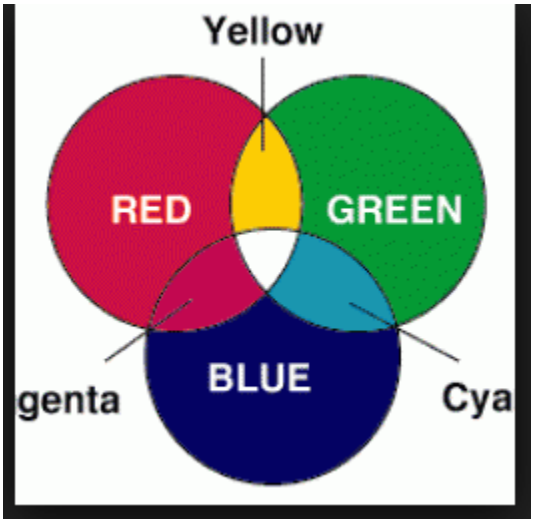
2. Grayscale image

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.



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.

3.0 Methodology

3.1 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

3.1.1 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')
```

3.1.2 The type and shape of the array.

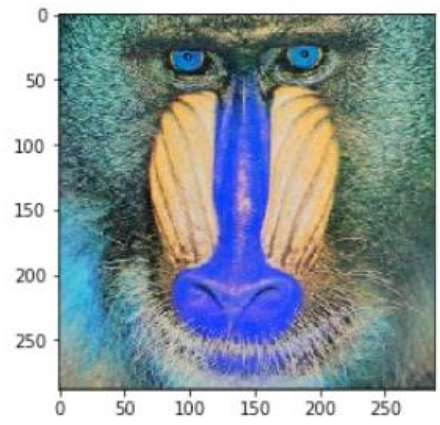
```
type(img_raw)
numpy.ndarray
```

```
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.

3.1.3 viewing the image

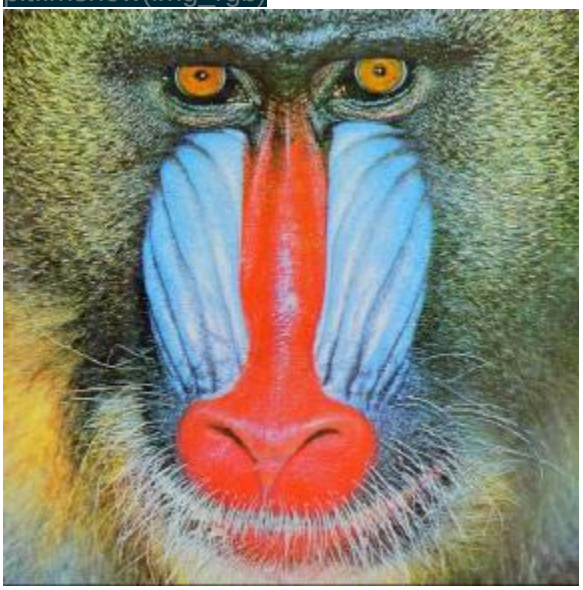
```
plt.imshow(img_raw)
```



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
```

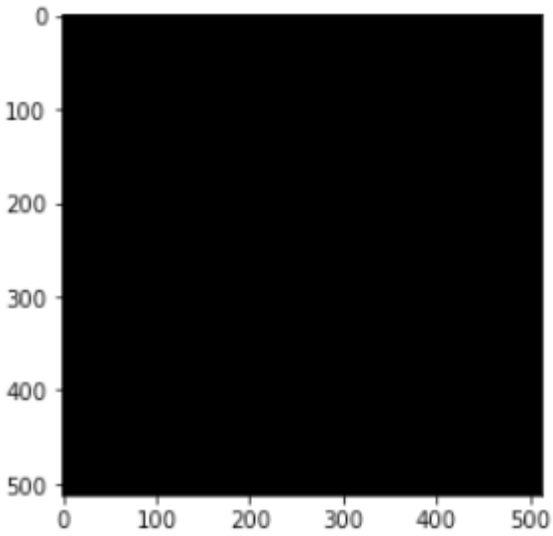
Create a black image which will act as a template.

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

Display the black image.

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

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



3.2 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)

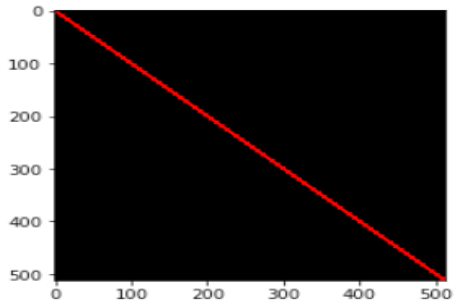
Color: The color of the shape that is to be drawn. It is passed as a tuple, eg: `(255,0,0)`. For grayscale, it will be the scale of brightness.

The thickness of the geometrical figure.

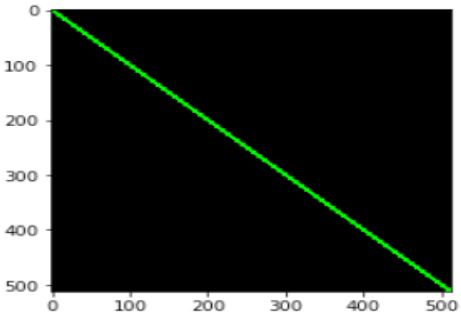
3.2.1 Straight Line

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)
```



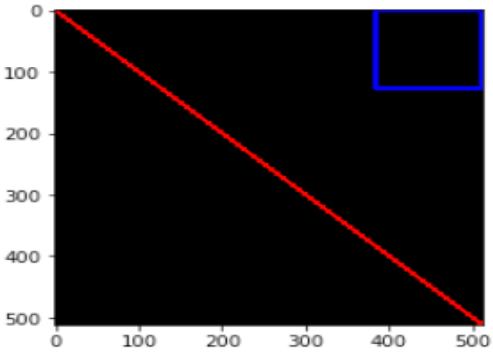

```
# Draw a diagonal green line with thickness of 5 px
line_green = cv2.line(img,(0,0),(511,511),(0,255,0),5)
plt.imshow(line_green)
```



3.2.2 Rectangle

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

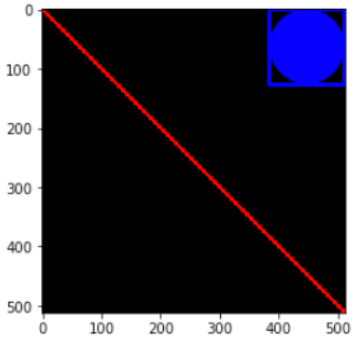
```
#Draw a blue rectangle with a thickness of 5 px

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



3.2.3 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:

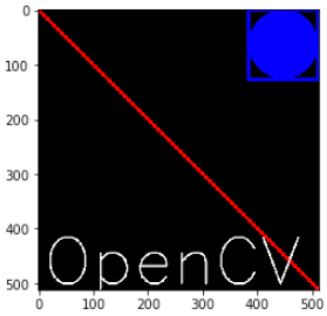
Text to be written

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.

3.2.4 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 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:

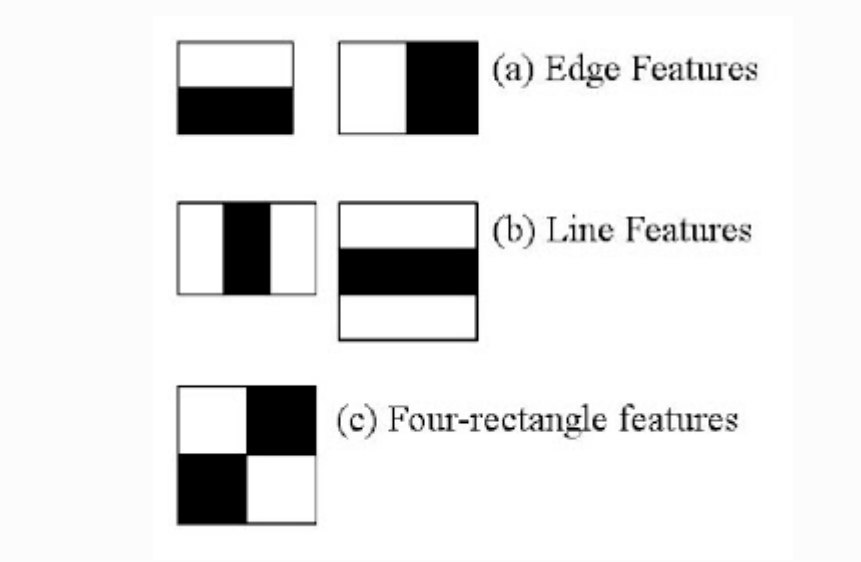
Haar Classifier and

Local Binary Pattern(LBP) classifier.

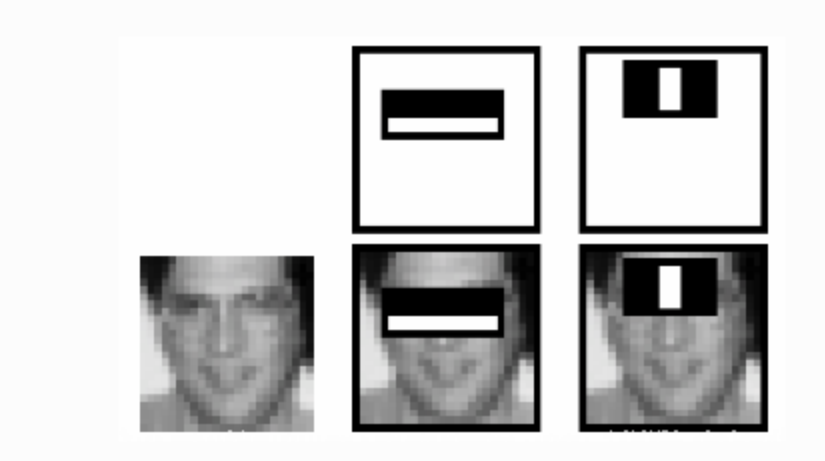
3.3 Let us now try and understand how the algorithm works on images in steps:

3.3.1 '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.

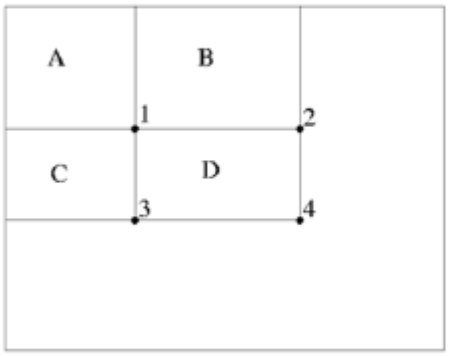


For demonstration purpose, let's say we are only extracting two features, hence we have only two windows here. The first feature relies on the point that the eye region is darker than the adjacent cheeks and nose region. The second feature focuses on the fact that eyes are kind of darker as compared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

3.3.2 '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.

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

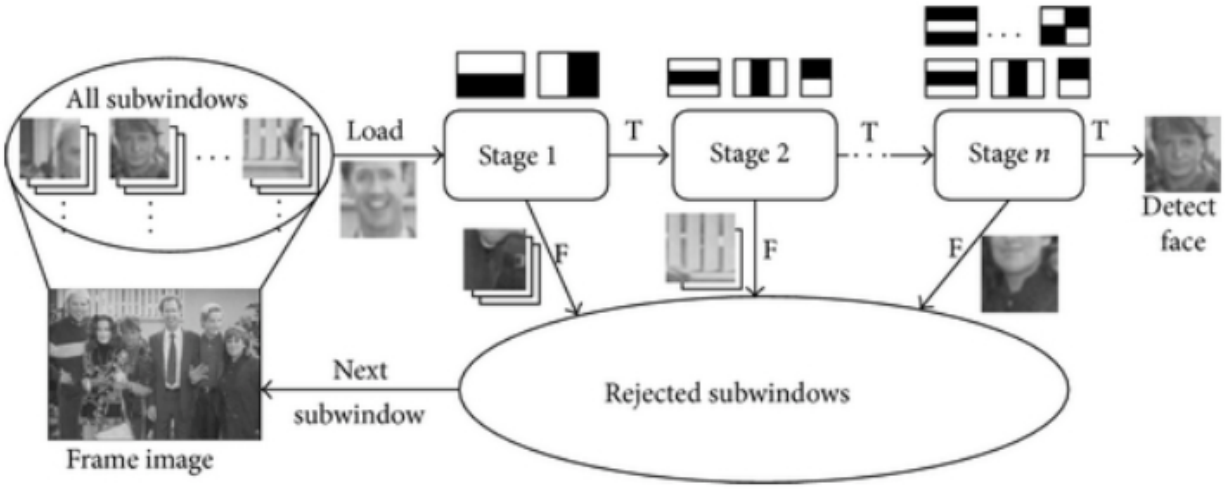
15 + 16 + 14 + 28 + 27 + 11 =
101 + 450 – 254 – 186 = 111

3.3.3 '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.

3.3.4 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.The Paul- Viola algorithm can be visualized as follows

3.4 Face Detection with OpenCV-Python

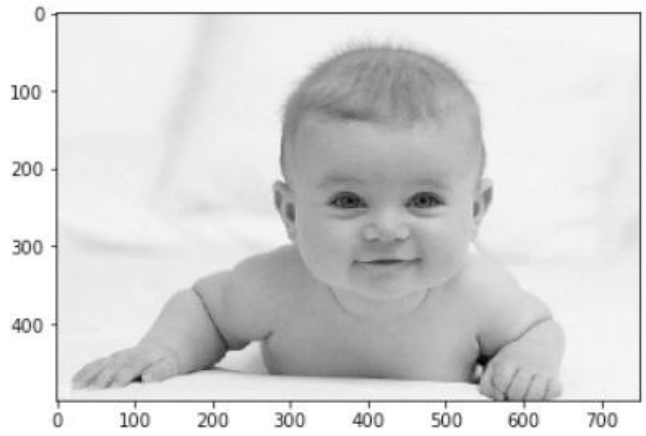
Now we have a fair idea about the intuition and the process behind Face recognition. Let us now use OpenCV library to detect faces in an image.

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 . 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')
```

3.4.1 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 .

```
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
```

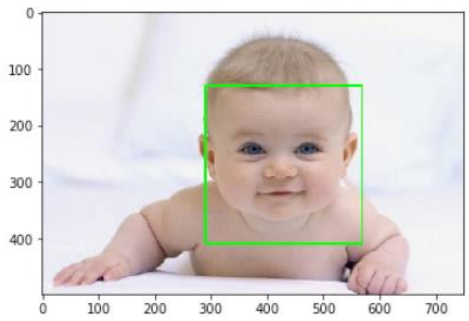
Our next step is to loop over all the coordinates it returned and draw rectangles around them using Open CV. We will be drawing a green rectangle with a thickness of 2

```
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

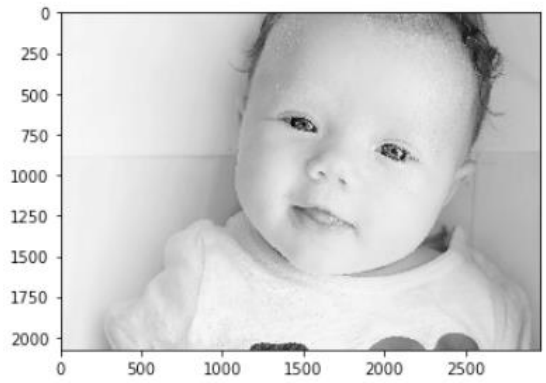
```
def detect_faces(cascade, test_image, scaleFactor = 1.1):
    # create a copy of the image to prevent any changes to the original one.
    image_copy = test_image.copy()

    #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')
```

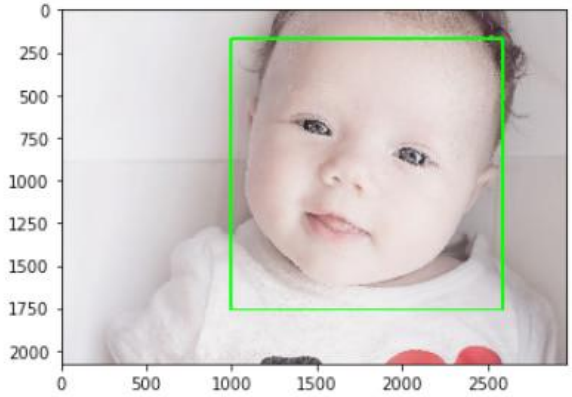
```
<matplotlib.image.AxesImage at 0x11f065be0>
```




```
#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

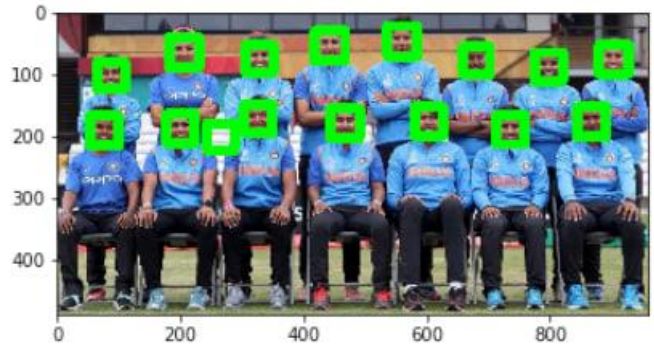
Let us now see if the function works well on a group photograph or not. We shall be using the picture below for our purpose.

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))
```



4.0 CHALLENGES

The main purpose is to recognize a specific object in real time from a large number of objects. Most recognition systems [2] are poorly scalable with many recognizable objects. Computational cost rises as the number of objects increases. Comparing and querying images using color, texture, and shape are not enough because two objects might have same attributes. Designing a recognition system with abilities to work in the dynamic environment and behave like a human is difficult. Some main challenges to design object recognition system are lighting, dynamic background, the presence of shadow, the motion of the camera, the speed of the moving objects, and intermittent object motion weather conditions etc.

5.0 CONCLUSION

The possibilities of using computer vision to solve real world problems are immense. The basics of object detection along with various ways of achieving it and its scope has been discussed. Python has been preferred over MATLAB for integrating with OpenCV because when a Matlab program is run on a computer, it gets busy trying to interpret all that Matlab code as Matlab code is built on Java. OpenCV is basically a library of functions written in C\C++. Additionally, OpenCV is easier to use for someone with little programming background. So, it is better to start researching on any concept of object detection using OpenCV-Python. Feature understanding and matching are the major steps in object detection and should be performed well and with high accuracy. Deep Face is the most effective face detection method that is preferred over Haar-Cascade by most of the social applications like facebook, snap chat, Instagram etc. In the coming days, OpenCV will be immensely popular among the coders and will also be the prime requirement of IT companies. To improve the performance of object detection IOU measures are used.

6.0 REFERENCES

[1] Khushboo Khurana and Reetu Awasthi,"Techniques for Object Recognition in Images and Multi-Object Detection",(IJARCET), ISSN:2278-1323,4th, April 2013.
[2] Latharani T.R., M.Z. Kurian, Chidananda Murthy M.V,"Various Object Recognition Techniques for Computer Vision", Journal of Analysis and Computation, ISSN: 0973-2861.
[3] Md Atiqur Rahman and Yang Wang, "Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation," in Object detection, Department of Computer Science, University of Manitoba, Canada, 2015.
[4] Nidhi, "Image Processing and Object Detection", Dept. of Computer Applications, NIT, Kurukshetra, Haryana, 1(9): 396-399, 2015.
[5] R. Hussin, M. Rizon Juhari, Ng Wei Kang, R.C.Ismail, A.Kamarudin, "Digital Image Processing Techniques for Object Detection from Complex Background Image,"Perlis, Malaysia: School of Microelectronic Engineering, University Malaysia Perlis, 2012.
[6] S.Bindu, S.Prudhvi, G.Hemalatha, Mr.N.Raja Sekhar, Mr. V.Nanchariah,"Object Detection from Complex Background Image using Circular Hough Transform", IJERA, ISSN: 2248-9622, Vol. 4, Issue 4(Version 1), April 2014, pp.23-28.
[7] Shaikh, S.H; Saeed, K, and Chaki.N,"Moving Object Detection Using Background Subtraction" Springer, ISBN:978-3-319-07385-9.
[8] Shijian Tang and Ye Yuan,"Object Detection based on Conventional Neural Network".
[9] Opencv.org, „About OpenCV", 2017. [Online]. Available:http://www.opencv.org/about
[10]Numpy.org, 2017. [Online]. Available: http://www.numpy.org