

FACE DETECTION AND RECOGNITION

A Report submitted in partial fulfillment of the requirements for the
award of degree of

Bachelor of Technology

In

Electronics and Communication Engineering

Under the Supervision of:

Ms. DEEPTI DESHWAL

By:

SHUBHAM SINGHAL (05715002816)

ROHAN AGARWAL (04315002816)

MAYANK GUPTA (02715002816)



MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY

C-4, Janakpuri, New Delhi-58

Affiliated to Guru Govind Singh Indraprastha University, Delhi

December, 2019

ACKNOWLEDGEMENT

We are highly indebted to our project mentors **Ms. Deepti Deshwal** , Professor ECE , for her continuous support, Supervision guidance and motivation throughout the tenure of my project in spite of their hectic schedule who truly remained driving spirit in our project and her experience gave us light in handling research project and us in clarifying the abstract concepts, requiring knowledge and perception, handling critical situations and in understanding the objective of our work.

We express our sincere thanks and gratitude to **Dr. Puneet Azad** for providing us the opportunity to do the project which was of great interest to us.

At last we thank the almighty, who had given the strength to complete this project on time

Finally, we would like to express my deep appreciation to our family and friends who have been a constant source of inspiration. we are internally grateful to them for always encouraging me wherever and whenever we needed them.

ROHAN AGARWAL (04315002816)

SHUBHAM SINGHAL (05715002816)

MAYANK GUPTA (02715002816)

CERTIFICATE

This is to certify that project work done on “FACE DETECTION AND RECOGNITION” submitted to Maharaja Surajmal Institute of Technology, Janakpuri Delhi by “Rohan Agarwal, Shubham Singhal, Mayank Gupta” in partial fulfilment of the requirement of the award of degree of Bachelors of Technology, is a bona fide work carried out by him/her under my supervision and guidance. This project work comprises of original work and has not been submitted anywhere else for any other degree to best of my knowledge.

Ms. Deepti Deshwal
(Project Supervisor)

Dr. Puneet Azad
(HOD,ECE)

DECLARATION

We, student of B.Tech(Electronics and Communication Engineering) hereby declare that the project work done on “FACE DETECTION AND RECOGNITION” submitted to Maharaja Surajmal Institute of Technology, Janakpuri Delhi in partial fulfillment of the requirement for the award of degree of Bachelors of Technology comprises of our original work and has not been submitted anywhere else for any other degree to the best of our knowledge.

ROHAN AGARWAL

(04315002816)

SHUBHAM SINGHAL

(05715002816)

MAYANK GUPTA

(02715002816)

Contents Table

S. No.	Topic	Page No.
1)	List of Figures	I
2)	Abstract	III
3)	Chapter 1: Introduction	1
	1.1 History of Face Recognition	1
4)	Chapter 2: OpenCV-Python	2
	2.1 Overview	2
	2.2 Installation	2
	2.3 Images and arrays	3
	2.3.1 Binary Image	4
	2.3.2 Grayscale Image	4
	2.3.3 Colored Image	5
	2.4 Images and OpenCV	6
	2.4.1 Importing Images in OpenCV	6
	2.4.2 Saving Images	7
	2.5 Basic Operation on Images	7
	2.5.1 Drawing on Images	7
	2.5.2 Functions and Attributes 2.5.2.1 Straight line 2.5.2.2 Rectangle 2.5.2.3 Circle	8
	2.5.3 Writing on Images	11
5)	Chapter 3: Face Detection	12
	3.1 Overview	12
	3.2 Haar feature-based cascade classifiers	12
	3.2.1 Haar Based face detection	13

	3.3 Face Detection with OpenCV-Python	14
	3.3.1 Loading the classifier for frontal face	16
	3.3.2 Face detection	16
	3.3.3 Face Detection with generalized function	18
	3.3.4 Testing the function on a group image	20
6)	Chapter 4: Face Recognition	22
	4.1 Eigenface	22
	4.2 Fisherface	23
	4.3 Local Binary Pattern Histogram	24
7)	Chapter 5: Methodology	26
	5.1 Face recognition Process	26
	5.1.1 Collecting the Image Data	26
	5.1.2 Training the Classifiers	27
	5.1.3 The Face Recognition	29
8)	Results and Conclusions	31
	6.1 Results	31
	6.2 Conclusion	32
9)	References	33
10)	Code	34

List of Figures

Fig. No.	Description	Page No.
1.1	Features reading of face in latest face recognition technique	1
2.1	OpenCV Logo	2
2.2	Relation between bits/pixels and possible colors	3
2.3	'0' represents pure black and '1' represents pure white	4
2.4	Grayscale Image	4
2.5	Sample image of monkey	5
2.6	Sample Image in 3D array	6
2.7	A black Image with given resolution	8
2.8	A red Line on Black Box	9
2.9	Rectangle added to last image	10
2.10	Circle added to last image	10
2.11	Text added to previous Image	11
3.1	Some examples of Haar features	13
3.2	Several Haar features matched to image	14
3.3	Flow chart of Haar cascade	14
3.4	Sample Image Taken	15
3.5	Grayscale of sample image	16
3.6	Face surrounded by rectangle	18
3.7	New sample image	19
3.8	Grayscale of new sample	20
3.9	Face detected using Function	20
3.10	The Indian Women's Cricket Team	21
3.11	Faces detected in group image	21
4.1	Pixels of the image are reordered to perform calculations for Eigenface	22
4.2	The first component of PCA and LDA. Classes in PCA looks more mixed than of LDA	24

4.3	Local binary pattern histogram generating 8-bit number	24
4.4	The results are same even if brightness is changed	25
5.1	The Flowchart for the image collection	27
5.2	Flowchart of the training application	29
5.3	Flow chart of face recognition application	30
6.1	Test for Priyanka.jpg	31
6.2	Test for Kangana.jpg	31
6.3	Test for Webcam video	32

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. OpenCV has three built in face recognizers and thanks to OpenCV's clean coding, you can use any of them by just changing a single line of code. There are two kinds of methods that are currently popular in developed face recognition pattern namely, Eigenface method and Fisherface method and another one Local Binary Patterns Histograms (LBPH) Face Recognizer. Facial image recognition Eigenface method is based on the reduction of face- dimensional space using Principal Component Analysis (PCA) for facial features. The main purpose of the use of PCA on face recognition using Eigen faces was formed (face space) by finding the eigenvector corresponding to the largest eigenvalue of the face image. The area of this project face detection system with face recognition is Image processing.

Chapter 1: Introduction

Face recognition is the task of identifying an already detected object as a known or unknown face. Often the problem of face recognition is confused with the problem of face detection. Face Recognition on the other hand is to decide if the "face" is someone known, or unknown, using for this purpose a database of faces in order to validate this input face.

1.1 History of Face Recognition

Earlier facial recognition technology was considered as an idea of science fiction. But in the past decade, facial recognition technology has not only become real — but it's widespread. Today, people can easily read articles and news stories about facial recognition everywhere. Here is the history of facial recognition technology and some ideas about its bright future.

Facial recognition technology along with AI (Artificial Intelligence) and Deep Learning (DL) technology are benefiting several industries. These industries include law enforcement agencies, airports, mobile phone manufacturing companies, home appliance manufacturing companies, etc.

Nowadays even retailers are using AI-based facial recognition technology to prevent violence and crime. Airports are getting better-secured environment, mobile phone makers are using face recognition to bring the biometric security feature in the devices.

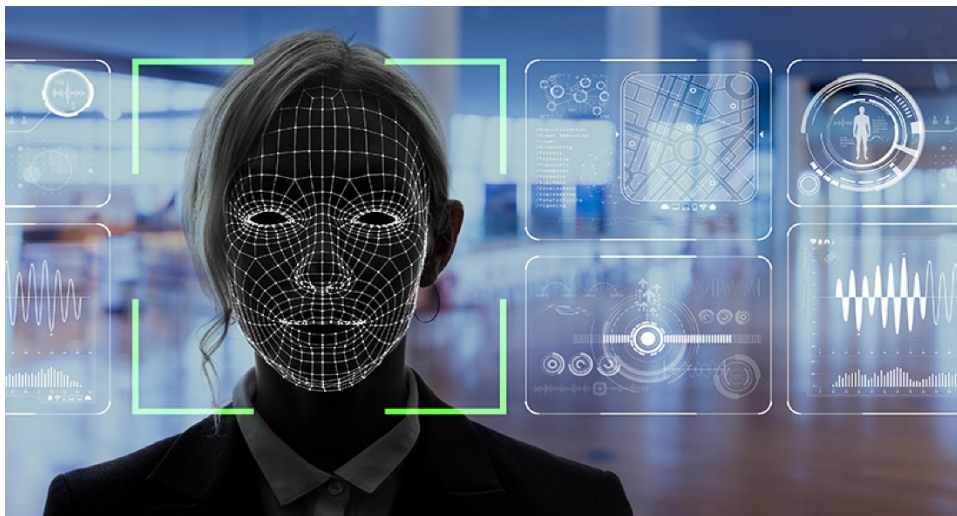


Fig1.1 Features reading of face in latest face recognition technique

Chapter 2: OpenCV-Python

2.1 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.



Fig 2.1 OpenCV Logo

2.2 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:

Please refer to the detailed documentation [here](#) for Windows and [here](#) for Mac.

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.

2.3 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	$2^{16} = 65000$

Fig2.2 Relation between bits/pixels and possible colors

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

2.3.1 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.

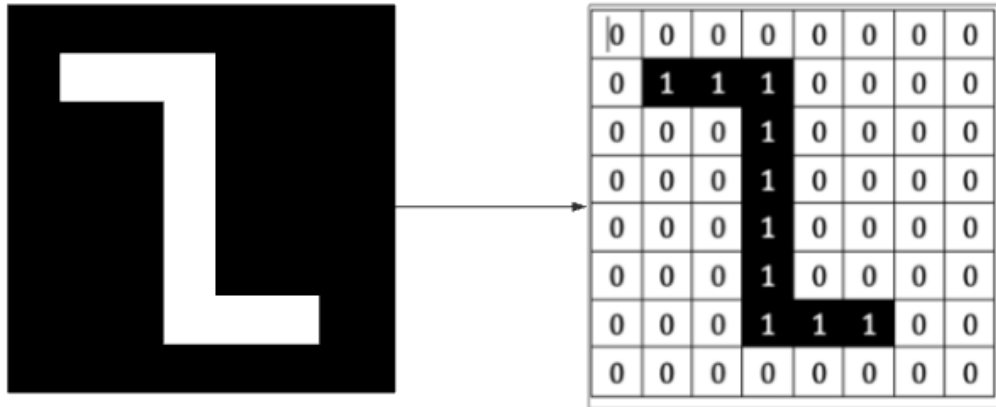


Fig 2.3 '0' represents pure black and '1' represents pure white

2.3.2 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.

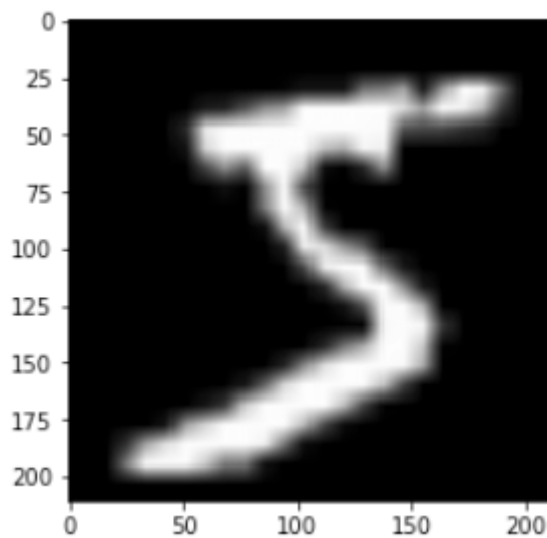


Fig 2.4 Grayscale Image

2.3.3 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.

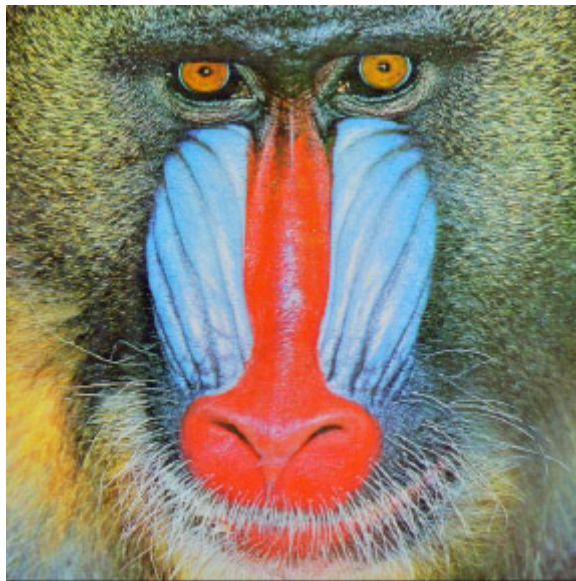


Fig 2.5 Sample image of monkey

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

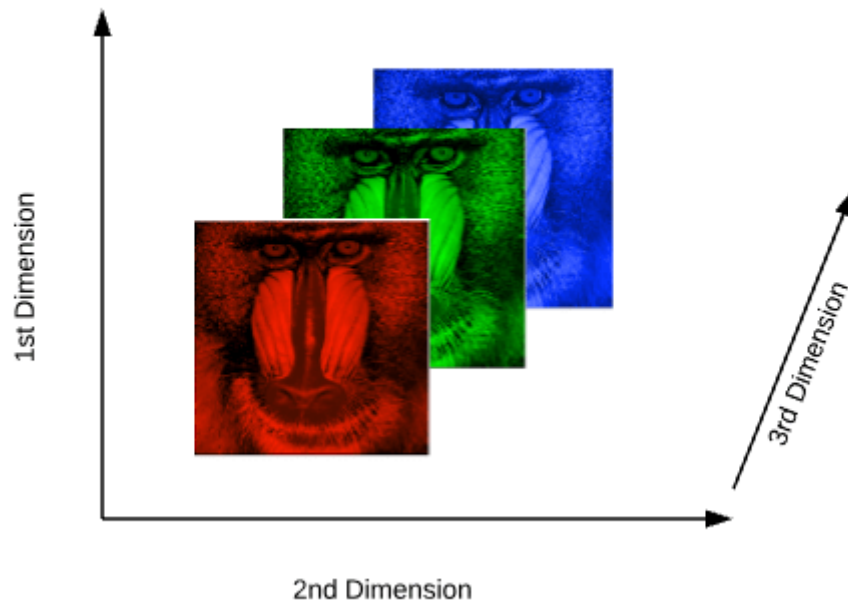


Fig 2.6 Sample Image in 3D array

2.4 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.

2.4.1 Importing Images in OpenCV

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 fulfil:

- 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.

2.4.2 Saving 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.

2.5 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.

2.5.1 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>
```

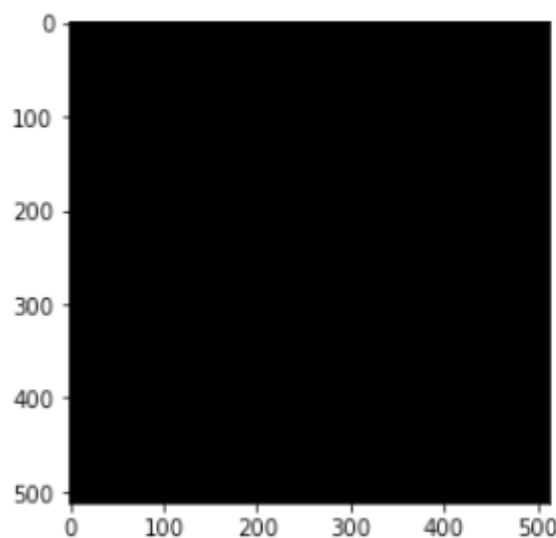


Fig 2.7 A black Image with given resolution

2.5.2 Functions and 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.

2.5.2.1 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)
```

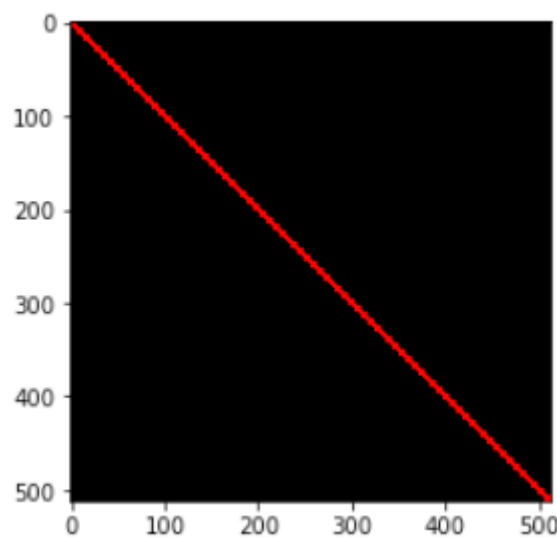


Fig 2.8 A red Line on Black Box

2.5.2.2 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)
```

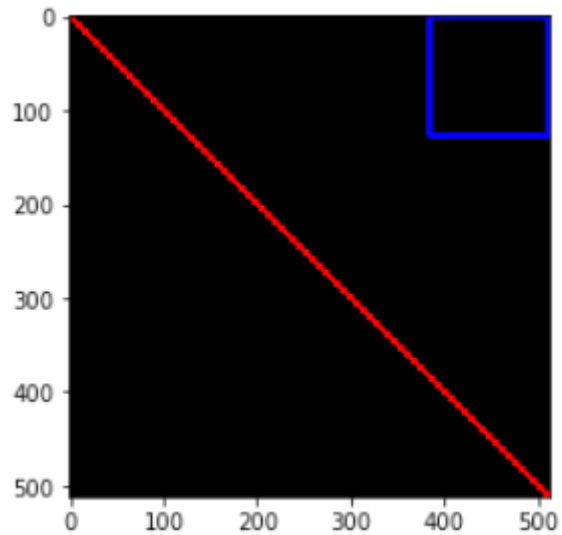


Fig 2.9 Rectangle added to last image

2.5.2.3 Circle

For a circle, we need to pass its centre 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)
```

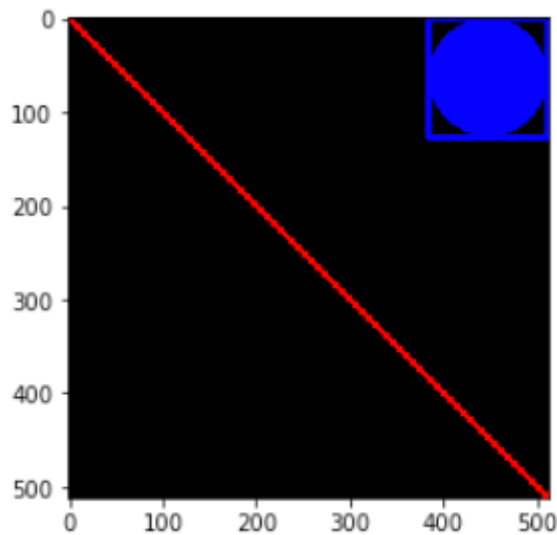


Fig 2.10 Circle added to last image

2.5.3 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)
```

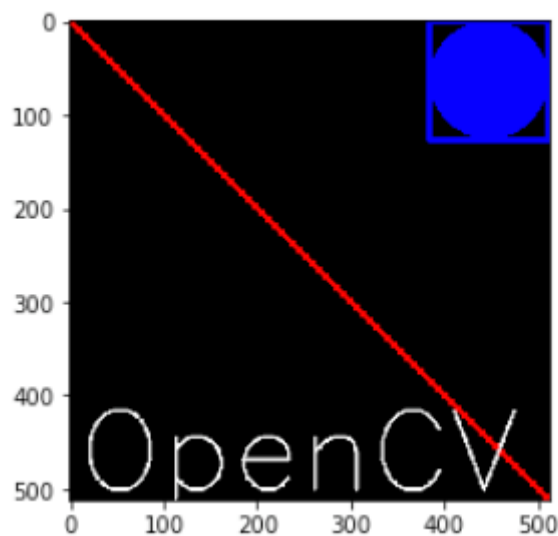


Fig 2.11 Text added to previous Image

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

Chapter 3: Face Detection

3.1 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
- Local Binary Pattern (LBP) classifier.

3.2 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.

3.2.1 Haar Based face detection

A Haar wavelet is a mathematical fiction that produces square-shaped waves with a beginning and an end and used to create box shaped patterns to recognise signals with sudden transformations. An example is shown in figure 3.1. By combining several wavelets, a cascade can be created that can identify edges, lines and circles with different colour intensities. These sets are used in Viola Jones face detection technique in 2001 and since then more patterns are introduced for object detection as shown in figure 3.1. To analyse an image using Haar cascades, a scale is selected smaller than the target image. It is then placed on the image, and the average of the values of pixels in each section is taken. If the difference between two values pass a given threshold, it is considered a match. Face detection on a human face is performed by matching a combination of different Haar-like-features. For example, forehead, eyebrows and eyes contrast as well as the nose with eyes as shown below in figure A single classifier is not accurate enough. Several classifiers are combined as to provide an accurate face detection system as shown in the block diagram below in figure 3.3.

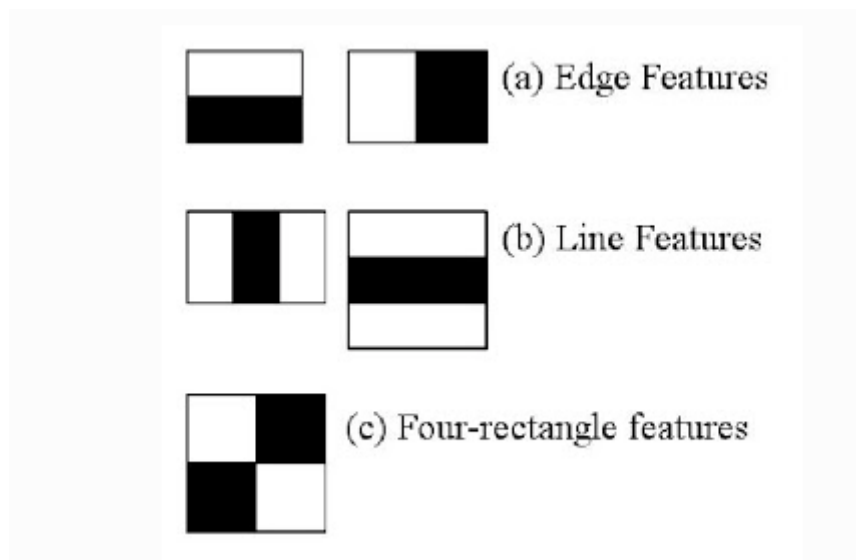


Fig 3.1 Some examples of Haar features

Figure 1: A Haar wavelet and resulting Haar-like features.in this project, a similar method is used effectively to by identifying faces and eyes in combination resulting

better face detection. Similarly, in viola Jones method, several classifiers were combined to create stronger classifiers. ADA boost is a machine learning algorithm that tests out several weak classifiers on a selected location and choose the most suitable. It can also reverse the direction of the classifier and get better results if necessary, Furthermore, Weight-update-steps can be updated



Fig 3.2 Several Haar features matched to image

only on misses to get better performance. The cascade is scaled by 1.25 and re-iterated in order to find different sized faces. Running the cascade on an image using conventional loops takes a large amount of computing power and time. Viola Jones used a summed area table (an integral image) to compute the matches fast. First developed in 1984, it became popular after 2001 when Viola Jones implemented Haar-cascades for face detection. Using an integral image enables matching features with a single Passover the image.

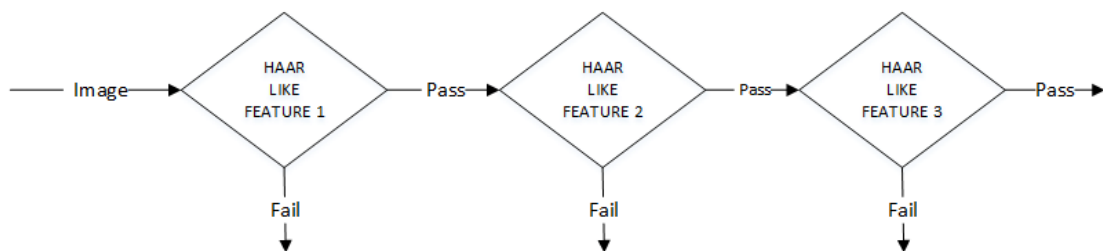


Fig 3.3 Flow chart of Haar cascade

3.3 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:



Fig 3.4 Sample Image Taken

```
#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 RGB format to be able to display its true colors. Let us write a small function for that.

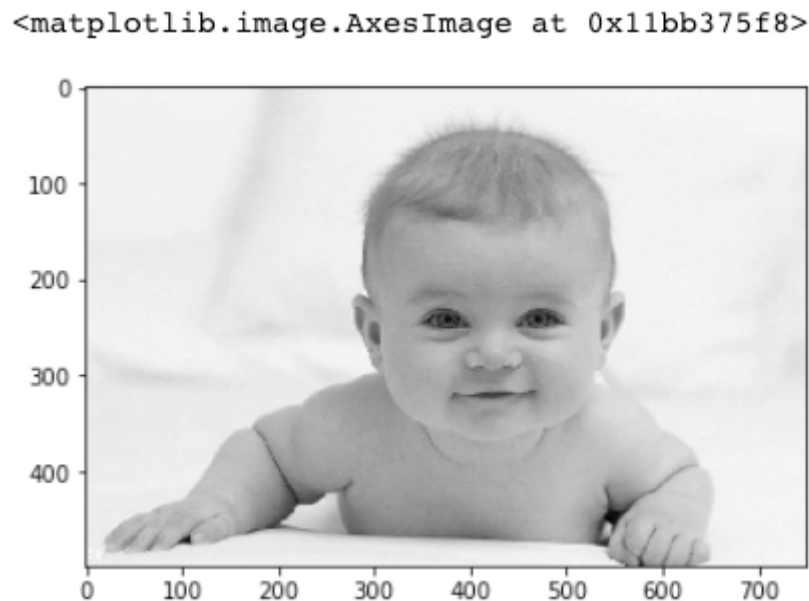


Fig 3.5 Grayscale of sample image

Since we know that OpenCV loads an image in BGR format, so we need to convert it into RGB 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)
```

3.3.1 Loading the classifier for frontal face

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

3.3.2 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 neighbours a rectangle should have to be called a face.

```
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>
```

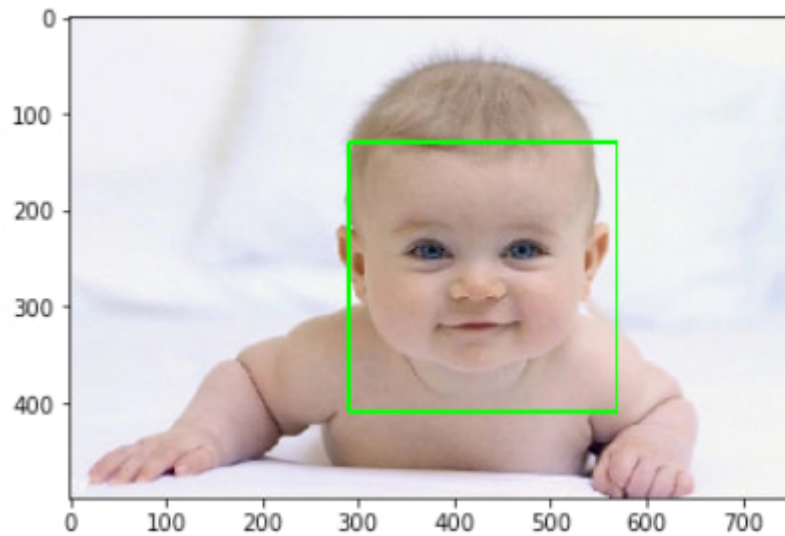


Fig 3.6 Face surrounded by rectangle

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.

3.3.3 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:



Fig 3.7 New sample image

```
#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>
```

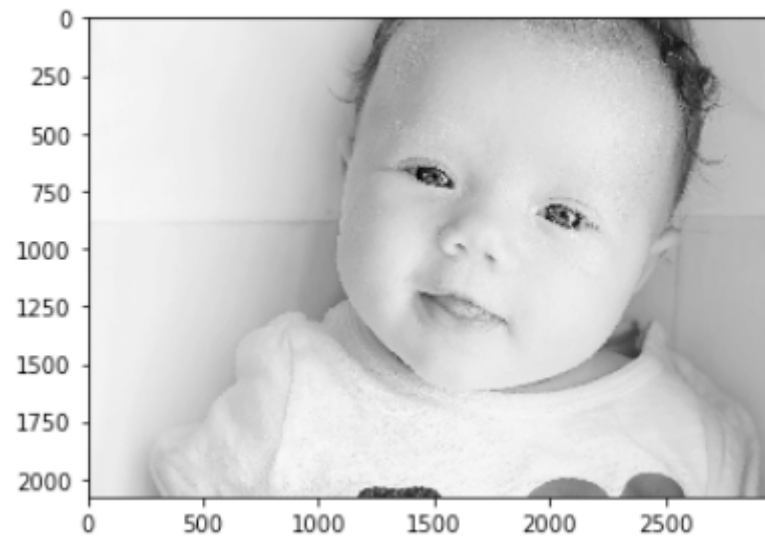


Fig 3.8 Grayscale of new sample

```
#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>
```

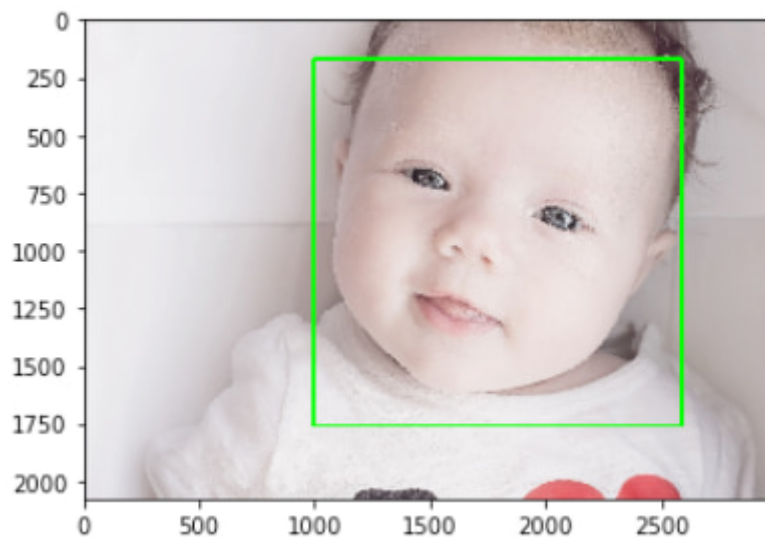


Fig 3.9 Face detected using Function

3.3.4 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.



Fig 3.10 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))
```



Fig 3.11 Faces detected in group image

Chapter 4: Face Recognition

The following sections describe the face recognition algorithms Eigenface, Fisherface, Local binary pattern histogram and how they are implemented in OpenCV

4.1 Eigenface

Eigenface is based on PCA that classify images to extract features using a set of images. It is important that the images are in the same lighting condition and the eyes match in each image. Also, images used in this method must contain the same number of pixels and in grayscale. For this example, consider an image with $n \times n$ pixels as shown in figure 4.1. Each row is concatenated to create a vector, resulting a $1 \times n^2$ matrix. All the images in the dataset are stored in a single matrix resulting a matrix with columns corresponding the number of images. The matrix is averaged (normalised) to get an average human face. By subtracting the average face from each image vector unique features to each face are computed. In the resulting matrix, each column is a representation of the difference each face has to the average human face. A simplified illustration can be seen in figure 4.1.

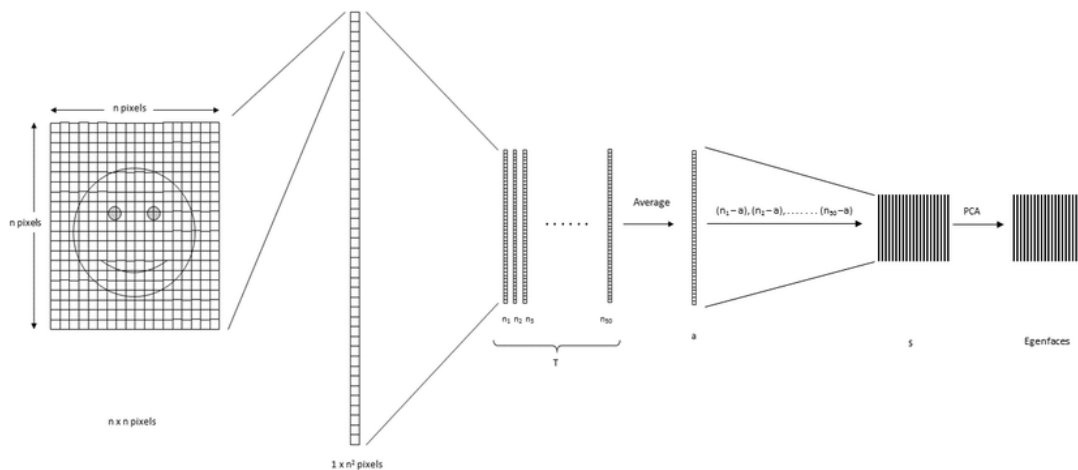


Fig 4.1 Pixels of the image are reordered to perform calculations for Eigenface

The next step is computing the covariance matrix from the result. To obtain the Eigen vectors from the data, Eigen analysis is performed using principal component analysis. From the result, where covariance matrix is diagonal, where it has the highest variance

is considered the 1st Eigen vector. 2nd Eigen vector is the direction of the next highest variance, and it is in 90 degrees to the 1st vector. 3rd will be the next highest variation, and so on. Each column is considered an image and visualised, resembles a face and called Eigenfaces. When a face is required to be recognised, the image is imported, resized to match the same dimensions of the test data as mentioned above. By projecting extracted features on to each of the Eigenfaces, weights can be calculated. These weights correspond to the similarity of the features extracted from the different image sets in the dataset to the features extracted from the input image. The input image can be identified as a face by comparing with the whole dataset. By comparing with each subset, the image can be identified as to which person it belongs to. By applying a threshold detection and identification can be controlled to eliminate false detection and recognition. PCA is sensitive to large numbers and assumes that the subspace is linear. If the same face is analysed under different lighting conditions, it will mix the values when distribution is calculated and cannot be effectively classified. This makes to different lighting conditions poses a problem in matching the features as they can change dramatically.

4.2 Fisherface

Fisherface technique builds upon the Eigenface and is based on LDA derived from Ronald Fishers' linear discriminant technique used for pattern recognition. However, it uses labels for classes as well as datapoint information. When reducing dimensions, PCA looks at the greatest variance, while LDA, using labels, looks at an interesting dimension such that, when you project to that dimension you maximise the difference between the mean of the classes normalised by their variance. LDA maximises the ratio of the between-class scatter and within-class scatter matrices. Due to this, different lighting conditions in images has a limited effect on the classification process using LDA technique. Eigenface maximises the variations while Fisherface maximises the mean distance between and different classes and minimises variation within classes. This enables LDA to differentiate between feature classes better than PCA and can be observed in figure 4.2. Furthermore, it takes less amount of space and is the fastest algorithm in this project. Because of these PCA is more suitable for representation of a set of data while LDA is suitable for classification.

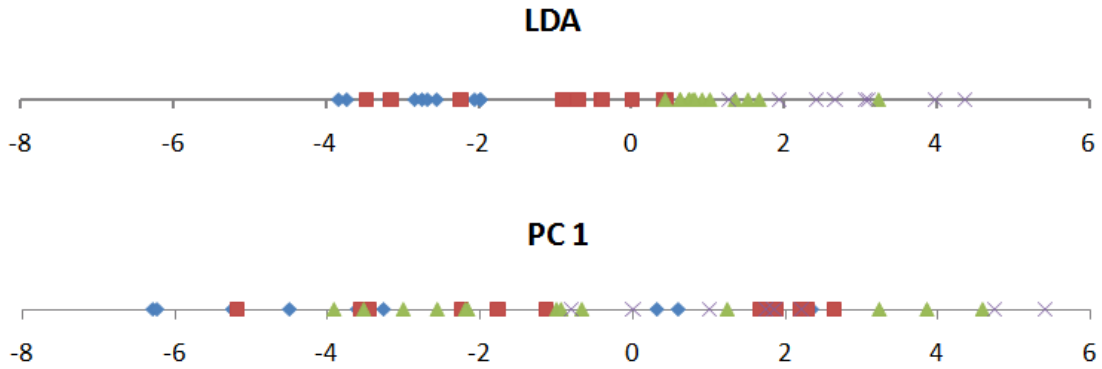


Fig 4.2 The first component of PCA and LDA. Classes in PCA looks more mixed than of LDA

4.3 Local Binary Pattern Histogram

Local binary patterns were proposed as classifiers in computer vision and in 1990 By Li Wang [4]. The combination of LBP with histogram-oriented gradients was introduced in 2009 that increased its performance in certain datasets. For feature encoding, the image is divided into cells (4 x 4 pixels). Using a clockwise or counter-clockwise direction surrounding pixel values are compared with the central as shown in figure 4.3. The value of intensity or luminosity of each neighbour is compared with the centre pixel. Depending if the difference is higher or lower than 0, a 1 or a 0 is assigned to the location. The result provides an 8-bit value to the cell. The advantage of this technique is even if the luminosity of the image

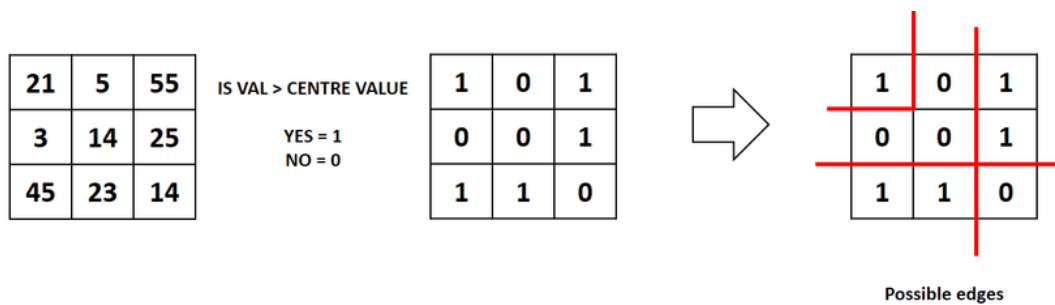


Fig 4.3 Local binary pattern histogram generating 8-bit number

is changed as in figure 4.4, the result is the same as before. Histograms are used in larger cells to find the frequency of occurrences of values making process faster. By analysing the results in the cell, edges can be detected as the values change. By computing the values of all cells and concatenating the histograms, feature vectors can be obtained. Images can be classified by processing with an ID attached. Input images are classified using the same process and compared with the dataset and distance is

obtained. by setting up a threshold, it can be identified if it is a known or unknown face. Eigenface and Fisherface compute the dominant features of the whole training set while LBPH analyse them individually.

Increase Brightness yet, same results

42	10	110	IS VAL > CENTRE VALUE YES = 1 NO = 0	1	0	1
6	28	50		0	0	1
90	46	28		1	1	0

Fig 4.4 The results are same even if brightness is changed

Chapter 5: Methodology

Below are the methodology and descriptions of the applications used for data gathering, face detection, training and face recognition. The project was coded in Python using a mixture of IDLE and PyCharm Ides.

5.1 Face recognition Process

For this project three algorithms are implemented independently. These are Eigenface, Fisherface and Linear binary pattern histograms respectively. All three can be implemented using OpenCV libraries. There are three stages for the face recognition as follows:

1. Collecting images IDs
2. Extracting unique features, classifying them and storing in XML files
3. Matching features of an input image to the features in the saved XML files and predict identity.

5.1.1 Collecting the Image Data

Collecting classification images is usually done manually using a photo editing software to crop and resize photos. Furthermore, PCA and LDA requires the same number of pixels in all the images for the correct operation. This time consuming and a laborious task is automated through an application to collect 50 images with different expressions. The application detects suitable expressions between 300ms, straightens any existing tilt and save them. The Flow chart for the application is shown in figure 5.1.

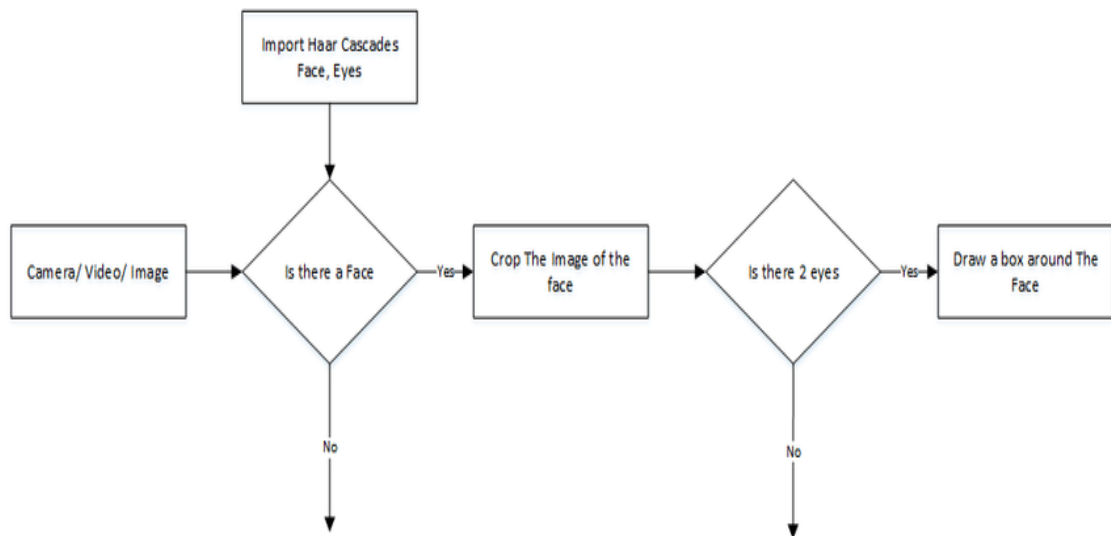


Fig 5.1 The Flowchart for the image collection

5.2.1 Collecting the image data Collecting classification images is usually done manually using a photo editing software to crop and resize photos. Furthermore, PCA and LDA requires the same number of pixels in all the images for the correct operation. This time consuming and a laborious task is automated through an application to collect 50 images with different expressions. The application detects suitable expressions between 300ms, straightens any existing tilt and save them. The Flow chart for the application is shown in figure 9. Figure 9: The Flowchart for the image collection Application starts with a request for a name to be entered to be stored with the ID in a text file. The face detection system starts the first half. However, before the capturing begins, the application check for the brightness levels and will capture only if the face is well illuminated. Furthermore, after the face is detected, the position of the eyes is analysed. If the head is tilted, the application automatically corrects the orientation. These two additions were made considering the requirements for Eigenface algorithm. The Image is then cropped and saved using the ID as a filename to be identified later. A loop runs this program until 50 viable images are collected from the person. This application made data collection efficient.

5.1.2 Training the Classifiers

OpenCV enables the creation of XML files to store features extracted from datasets using the FaceRe-cognizer class. The stored images are imported, converted to grayscale and saved with IDs in two lists with same indexes. FaceRecognizer objects

are created using face recogniser class. Each recogniser can take in parameters that are described below:

`cv2.face.createEigenFaceRecognizer()`

1. Takes in the number of components for the PCA for crating Eigenfaces. OpenCV documentation mentions 80 can provide satisfactory reconstruction capabilities.
2. Takes in the threshold in recognising faces. If the distance to the likeliest Eigenface is above this threshold, the function will return a -1, that can be used state the face is unrecognisable6

`cv2.face.createFisherfaceRecognizer()`

1. The first argument is the number of components for the LDA for the creation of Fisherfaces. OpenCV mentions it to be kept 0 if uncertain.
2. Similar to Eigenface threshold. -1 if the threshold is passed.

`cv2.face.createLBPHFaceRecognizer()`

1. The radius from the centre pixel to build the local binary pattern.
2. The Number of sample points to build the pattern. Having a considerable number will slow down the computer.
3. The Number of Cells to be created in X axis.
4. The number of cells to be created in Y axis.
5. A threshold value similar to Eigenface and Fisherface. if the threshold is passed the object will return -1

Recogniser objects are created and images are imported, resized, converted into numpy arrays and stored in a vector. The ID of the image is gathered from splitting the file name, and stored in another vector. By using `FaceRecognizer.train(NumPy Image, ID)` all three of the objects are trained. It must be noted that resizing the images were required only for Eigenface and Fisherface, not for LBPH. Next, the configuration model is saved as an XML file using `FaceRecognizer.save(FileName)`. In this project, all three are trained and saved through one application for convenience. The flow chart for the trainer is shown in figure 5.2.

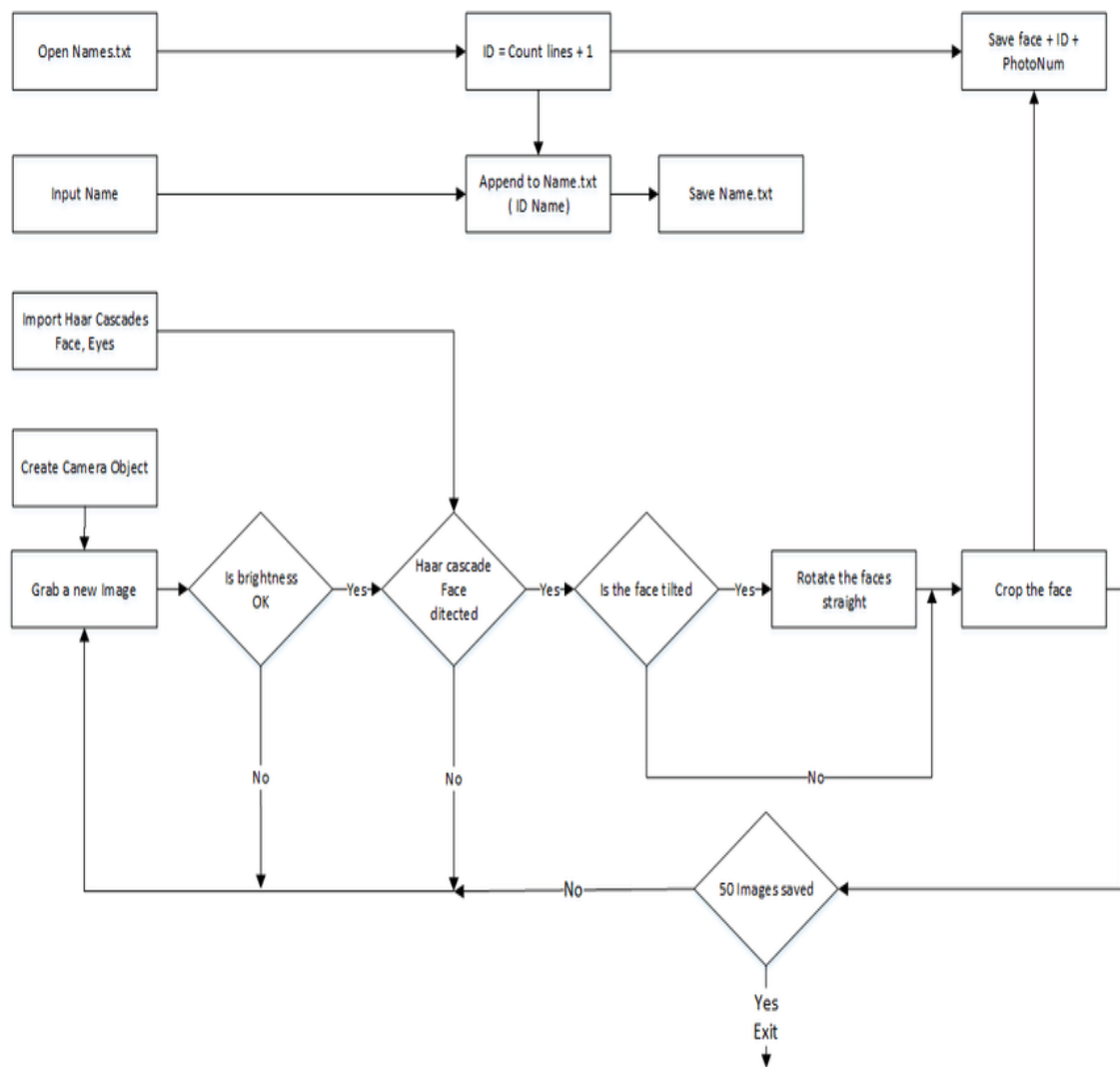


Fig 5.2 Flowchart of the training application

5.1.3 The Face Recognition

Face recogniser object is created using the desired parameters. Face detector is used to detect faces in the image, cropped and transferred to be recognised. This is done using the same technique used for the image capture application. For each face detected, a prediction is made using `FaceRecognizer.predict()` which return the ID of the class and confidence. The process is same for all algorithms and if the confidence his higher than the set threshold, ID is -1. Finally, names from the text file with IDs are used to display the name and confidence on the screen. If the ID is -1, the application will print unknown face without the confidence level. The flow chart for the application is shown in figure 5.3

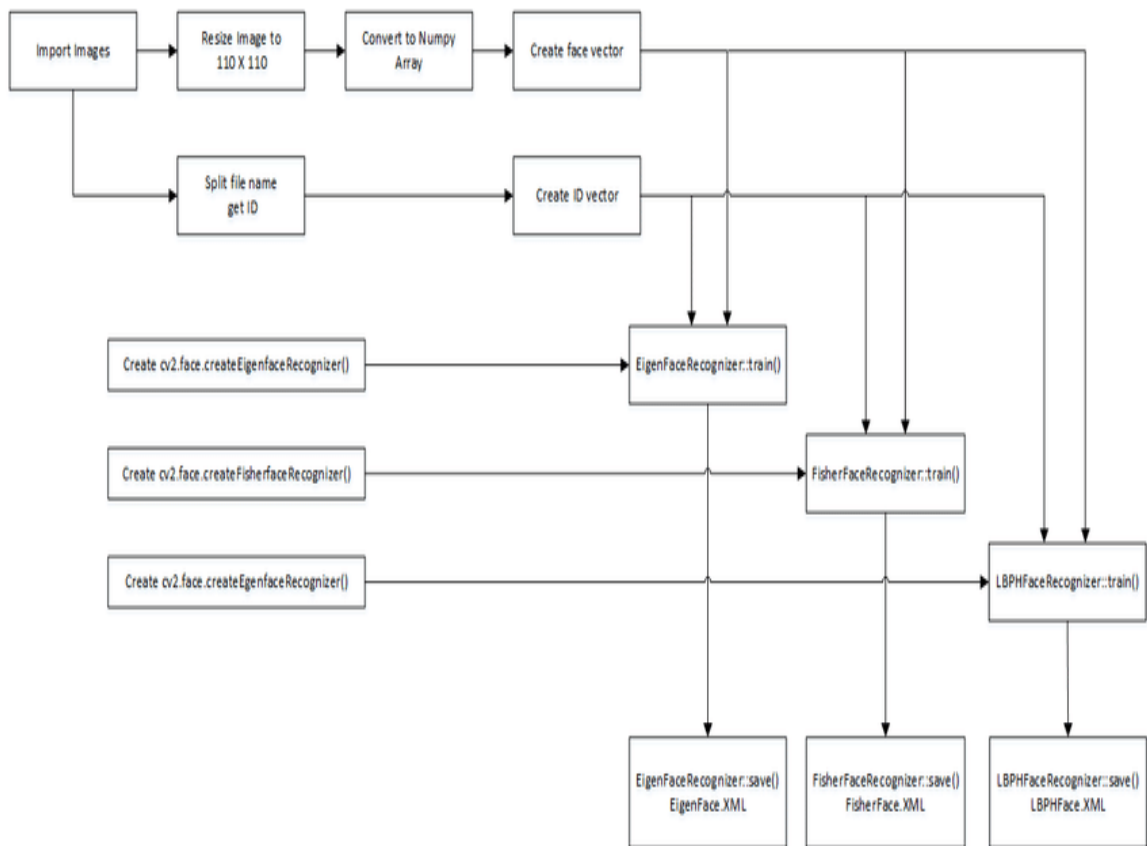


Fig 5.3 Flow chart of face recognition application

Results and Conclusion

6.1 Result

We saved images of Priyanka Chopra and Kangana Ranaut and Rohan agarwal in different subdirectories of train images and trained our code accordingly.

As we conduct the test on images and with webcam video, we obtained the following results:

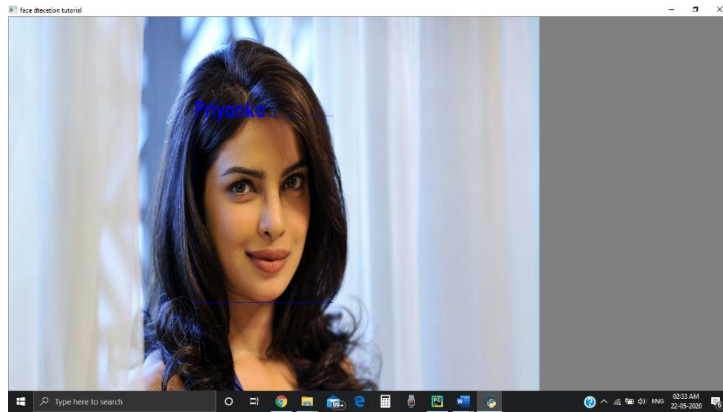


Fig 6.1 Test for Priyanka.jpg

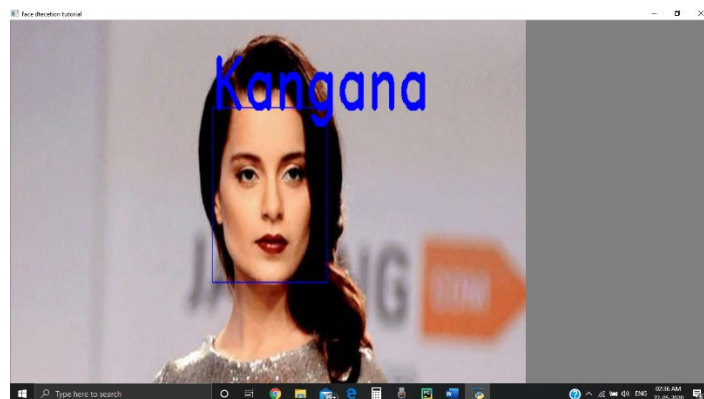


Fig 6.2 Test for Kangana.jpg

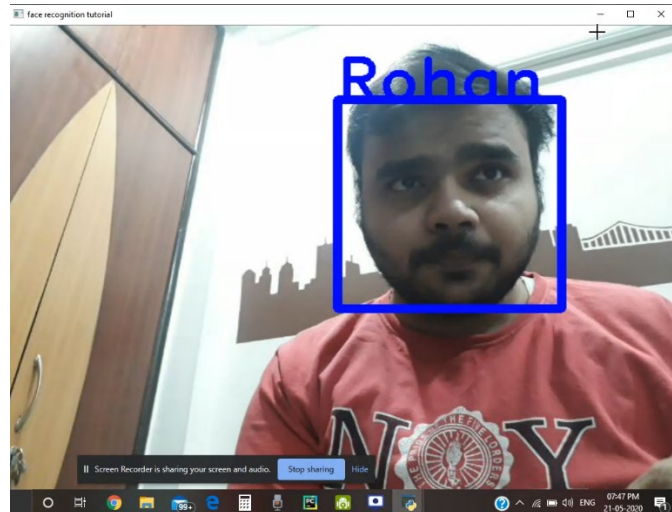


Fig 6.3 Test for Webcam video

6.2 Conclusion

This Report explains the technologies used in the project and the methodology used. Finally, it shows the results, discuss the challenges and how they were resolved followed by a discussion. Using Haar-cascades for face detection worked extremely well even when subjects wore spectacles. Real time video speed was satisfactory as well devoid of noticeable frame lag. Considering all factors, LBPH combined with Haar-cascades can be implemented as a cost effective face recognition platform. An example is a system to identify known troublemakers in a mall or a supermarket to provide the owner a warning to keep him alert or for automatic attendance taking in a class.

References

1. "OpenCV Guide", Seminar Presentation "Archived copy" (PDF). Archived from the original (PDF) on 2012-03-02. Retrieved 2011-08-02.
2. Prateek Jooshi, OpenCV with Python by Example, India, 2007, 120-158
3. SuperDataScience team, "Face recognition using OpenCV and Python: A beginner's guide", <https://www.superdatascience.com/blogs/opencv-face-recognition>, Aug 2017

Code

1. faceRecognition.py

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help FaceRecognition-master - faceRecognition.py - PyCharm
FaceRecognition-master faceRecognition.py
1 import cv2
2 import os
3 import numpy as np
4
5 #This module contains all common functions that are called in tester.py file
6
7
8 #Given an image below function returns rectangle for face detected alongwith gray scale image
9 def faceDetection(test_img):
10     gray_img=cv2.cvtColor(test_img,cv2.COLOR_BGR2GRAY)#convert color image to grayscale
11     face_haar_cascade=cv2.CascadeClassifier('HaarCascade/haarcascade_frontalface_default.xml')#Load haar classifier
12     faces=face_haar_cascade.detectMultiScale(gray_img,scaleFactor=1.32,minNeighbors=5)#detectMultiScale returns rectangles
13
14     return faces,gray_img
15
16 #Given a directory below function returns part of gray img which is face alongwith its label/ID
17 def labels_for_training_data(directory):
18     faces=[]
19     faceID=[]
20
21     for path,subdirname,filenames in os.walk(directory):
22         for filename in filenames:
23             if filename.startswith("."):
24                 print("Skipping system file")#Skipping files that start with .
25                 continue
26
27             id=os.path.basename(path)#fetching subdirectory names
28             img_path=os.path.join(path,filename)#fetching image path
```

```
25         continue
26
27         id=os.path.basename(path)#fetching subdirectory names
28         img_path=os.path.join(path,filename)#fetching image path
29         print("img_path:",img_path)
30         print("id:",id)
31         test_img=cv2.imread(img_path)#loading each image one by one
32         if test_img is None:
33             print("Image not loaded properly")
34             continue
35         faces_rect,gray_img=faceDetection(test_img)#Calling faceDetection function to return faces detected in particular image
36         if len(faces_rect)!=1:
37             continue #since we are assuming only single person images are being fed to classifier
38         (x,y,w,h)=faces_rect[0]
39         roi_gray=gray_img[y:y+w,x:x+h]#cropping region of interest i.e. face area from grayscale image
40         faces.append(roi_gray)
41         faceID.append(int(id))
42     return faces,faceID
43
44
45 #Below function trains haar classifier and takes faces,faceID returned by previous function as its arguments
46 def train_classifier(faces,faceID):
47     face_recognizer=cv2.Face_LBPHFaceRecognizer_create()
48     face_recognizer.train(faces,np.array(faceID))
49     return face_recognizer
50
51 #Below function draws bounding boxes around detected face in image
52 def draw_rect(test_img,faces):
53     labels_for_training_data(directory) for path,subdirname,filenames in os.walk(directory) for filename in filenames if filename.startswith(".")
54
55 Packages installed successfully: Installed packages: 'opencv-contrib-python' (yesterday 05:51 PM)
23:41 LF UTF-8 4 spaces Python 3.8 (untitled3)
```

```

49     return face_recognizer
50
51 #Below function draws bounding boxes around detected face in image
52 def draw_rect(test_img,face):
53     (x,y,w,h)=face
54     cv2.rectangle(test_img,(x,y),(x+w,y+h),(255,0,0),1)
55
56 #Below function writes name of person for detected label
57 def put_text(test_img,text,x,y):
58     cv2.putText(test_img,text,(x,y),cv2.FONT_HERSHEY_DUPLEX,2,(255,0,0),4)
59
60
61
62
63
64
65
66
67
68
69
70

```

labels_for_training_data() for path,subdirname,filenames... for filename in filenames if filename.startswith(".")

23:41 LF UTF-8 4 spaces Python 3.8 (untitled3)

2. tester.py

```

1 import cv2
2 import os
3 import numpy as np
4 import faceRecognition as fr
5
6
7 #This module takes images stored in disk and performs face recognition
8 test_img=cv2.imread('TestImages/Kangana.jpg')#test_img path
9 faces_detected,gray_img=fr.faceDetection(test_img)
10 print("faces_detected:",faces_detected)
11
12
13 #Comment below lines when running this program second time.Since it saves training.yml file in directory
14 #faces,faceID=fr.labels_for_training_data('trainingImages')
15 #face_recognizer=fr.train_classifier(faces,faceID)
16 #face_recognizer.write('trainingData.yml')
17
18
19 #Uncomment below line for subsequent runs
20 face_recognizer=cv2.face.LBPHFaceRecognizer_create()
21 face_recognizer.read('trainingData.yml')#use this to load training data for subsequent runs
22
23 names={0:"Priyanka",1:"Kangana",2:"Rohan"}#creating dictionary containing names for each label
24
25 for face in faces_detected:
26     (x,y,w,h)=face
27     roi_gray=gray_img[y:y+h,x:x+h]
28     label,confidence=face_recognizer.predict(roi_gray)#predicting the label of given image

```

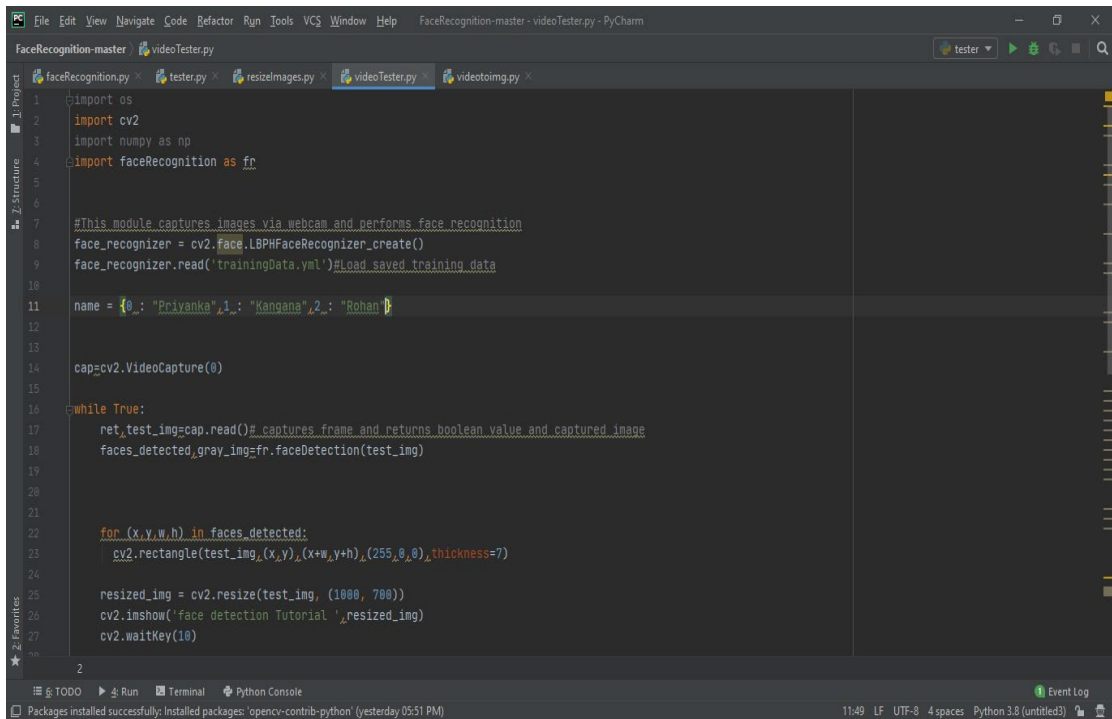
6:1 LF UTF-8 4 spaces Python 3.8 (untitled3)

```
25 for face in faces_detected:
26     (x,y,w,h)=face
27     roi_gray=gray_img[y:y+h,x:x+h]
28     label,confidence=face_recognizer.predict(roi_gray)#predicting the label of given image
29     print("confidence:",confidence)
30     print("label:",label)
31     fr.draw_rect(test_img,face)
32     predicted_name=name[label]
33     if(confidence>37):#if confidence more than 37 then don't print predicted face text on screen
34         continue
35     fr.put_text(test_img,predicted_name,x,y)
36
37 resized_img=cv2.resize(test_img,(1000,1000))
38 cv2.imshow("face detection tutorial",resized_img)
39 cv2.waitKey(0)#waits indefinitely until a key is pressed
40 cv2.destroyAllWindows
```

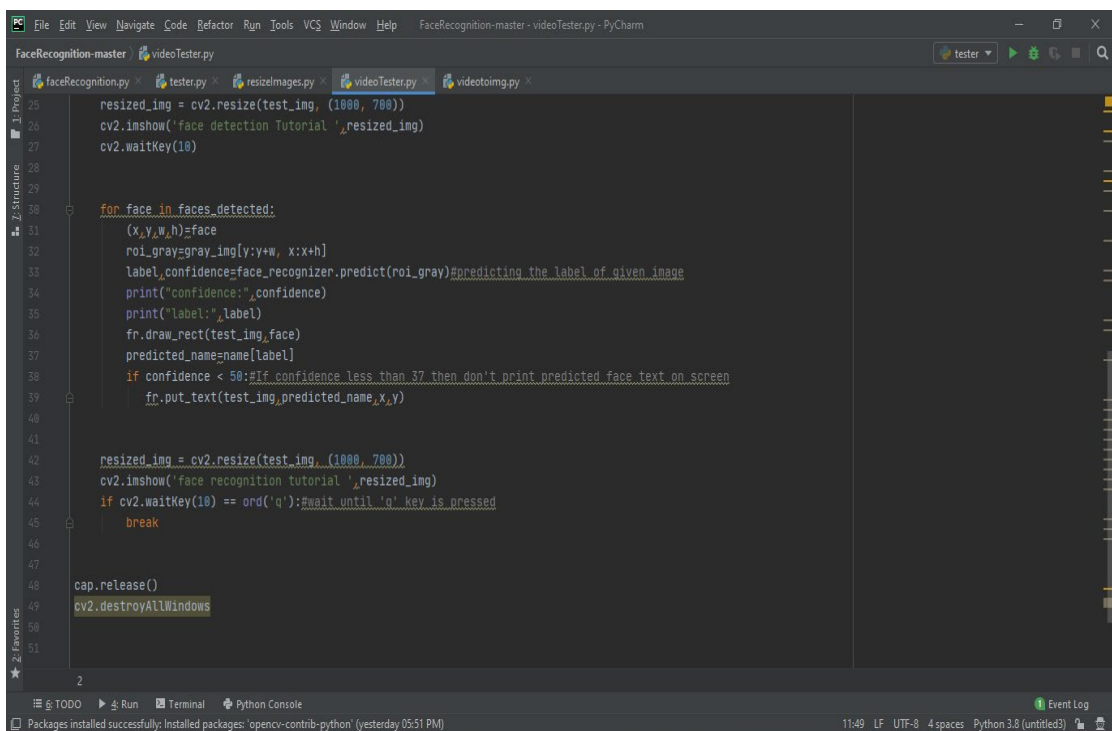
3. resizeimages.py

```
1 import cv2
2 import os
3 import numpy as np
4
5 #This module resizes image from a given directory to 100*100 pixels and writes all images to given directory
6 count=0
7
8 for path, subdirname, filenames in os.walk("trainingImages"):
9
10     for filename in filenames:
11         if filename.startswith("."):
12             print("Skipping File:",filename)#Skipping files that start with..
13             continue
14
15         img_path=os.path.join(path, filename)#fetching image path
16         print("img_path",img_path)
17         id=os.path.basename(path)#fetching subdirectory names
18         img = cv2.imread(img_path)
19         if img is None:
20             print("Image not loaded properly")
21             continue
22         resized_image = cv2.resize(img, (100, 100))
23         new_path="resizedTrainingImages"+"/"+str(id)
24         print("desired path is",os.path.join(new_path, "frame%d.jpg" % count))#write all images to resizedTrainingImages/id directory
25         cv2.imwrite(os.path.join(new_path, "frame%d.jpg" % count),resized_image)
26         count += 1
```

4. videoTester.py

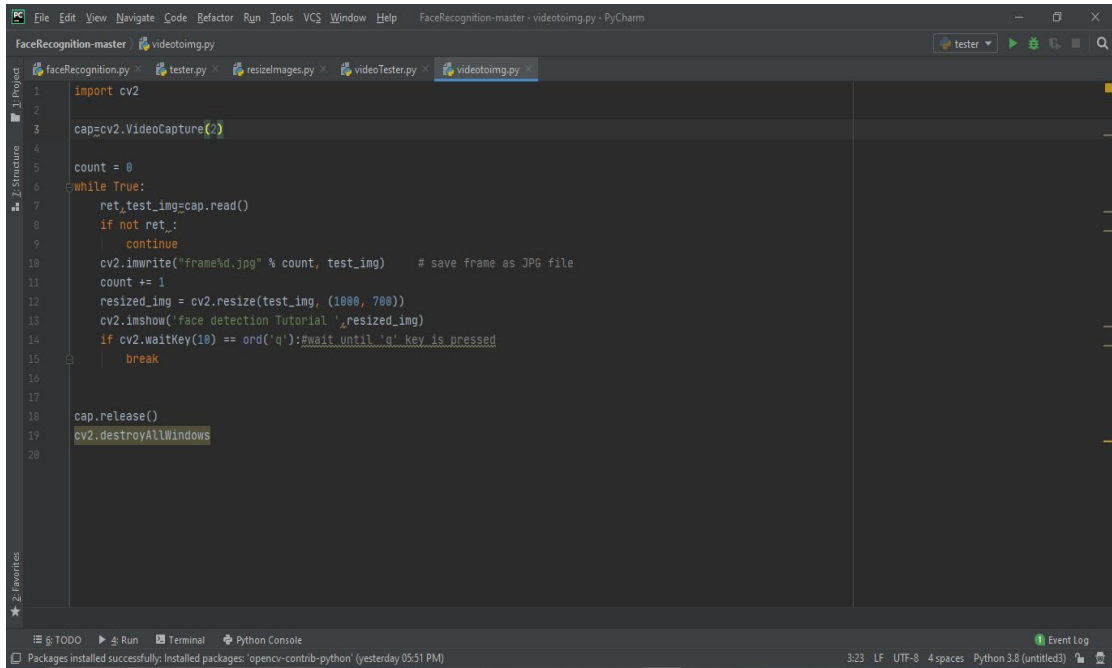


```
1 import os
2 import cv2
3 import numpy as np
4 import faceRecognition as fr
5
6
7 #This module captures images via webcam and performs face recognition
8 face_recognizer = cv2.face.LBPHFaceRecognizer_create()
9 face_recognizer.read('trainingData.yml') #load saved training data
10
11 name = {'0': "Priyanka", 1: "Kangana", 2: "Rohan"}
12
13
14 cap=cv2.VideoCapture(0)
15
16 while True:
17     ret,test_img=cap.read()# captures frame and returns boolean value and captured image
18     faces_detected_gray_img=fr.faceDetection(test_img)
19
20
21
22     for (x,y,w,h) in faces_detected:
23         cv2.rectangle(test_img,(x,y),(x+w,y+h),(255,0,0),thickness=7)
24
25     resized_img = cv2.resize(test_img, (1000, 700))
26     cv2.imshow('face detection Tutorial ',resized_img)
27     cv2.waitKey(10)
```



```
25     resized_img = cv2.resize(test_img, (1000, 700))
26     cv2.imshow('face detection Tutorial ',resized_img)
27     cv2.waitKey(10)
28
29     for face in faces_detected:
30         (x,y,w,h)=face
31         roi_gray=gray_img[y:y+w, x:x+h]
32         label_confidence=face_recognizer.predict(roi_gray)#predicting the label of given image
33         print("confidence:",confidence)
34         print("label:",label)
35         fr.draw_rect(test_img,face)
36         predicted_name=name[label]
37         if confidence < 50:#If confidence less than 37 then don't print predicted face text on screen
38             fr.put_text(test_img,predicted_name,x,y)
39
40
41     resized_img = cv2.resize(test_img, (1000, 700))
42     cv2.imshow('face recognition tutorial ',resized_img)
43     if cv2.waitKey(10) == ord('q'):#wait until 'q' key is pressed
44         break
45
46
47 cap.release()
48 cv2.destroyAllWindows
```

5. videotointg.py



```
1 import cv2
2
3 cap=cv2.VideoCapture(2)
4
5 count = 0
6 while True:
7     ret,test_img=cap.read()
8     if not ret:
9         continue
10    cv2.imwrite("frame%d.jpg" % count, test_img)    # save frame as JPG file
11    count += 1
12    resized_img = cv2.resize(test_img, (1000, 700))
13    cv2.imshow('face detection Tutorial',resized_img)
14    if cv2.waitKey(10) == ord('q'):wait_until 'q' key is pressed
15        break
16
17
18 cap.release()
19 cv2.destroyAllWindows
20
```

FaceRecognition-master - videotointg.py - PyCharm

faceRecognition.py × tester.py × resizeImages.py × videoTester.py × videotointg.py ×

1. Project

2. Favorites

3. TODO

4. Run

5. Terminal

6. Python Console

7. Packages installed successfully: Installed packages: 'opencv-contrib-python' (yesterday 05:51 PM)

3:23 LF UTF-8 4 spaces Python 3.8 (untitled3)

Event Log