FACE DETECTION & FACE RECOGNITION with IMAGE ENHANCEMENT

J Component Project Report for
CS4019- IMAGE PROCESSING
Slot – G2

Bachelor of Technology
in
ECE with Specialization in Internet of Things and Sensors

By

Name	Reg. No.	Phone	Email
Aditya Vikram	*16BIS0004*	*7004319715*	*Avikrama210@gmail.com*
Wriddhirup Dutta	*16BIS0145*	*70104470881*	*wriddhirup.dutta2016@vitstudent.ac.in*
Soumik Mitra	*16BIS0095*	*8072937470*	*soumik.mitra2016@vitstudent.ac.in*

Under the guidance of

Dr. NATARAJAN P.

School of Computer Science and Engineering

Vellore Institute of Technology, Vellore-632014

FALL 2018-19

CERTIFICATE

This is to certify that the project work entitled "***Face Detection & Face Recognition With Image Enhancement***" that is being submitted by ***Aditya Vikram, Wriddhirup Dutta, Soumik Mitra*** for **Image Processing** (CSE4019) is a record of bonafide work done under my supervision. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted for any other CAL course.

Place: Vellore

Date : 29.10.2018

Signature of Students:

ADITYA VIKRAM

SOUMIK MITRA

WRIDDHIRUP DUTTA

Signature of Faculty: _____

Natarajan P.

ABSTRACT

Identifying a person with an image has been popularized through the mass media. However, it is less robust to fingerprint or retina scanning. This report describes the face detection and recognition mini-project undertaken for the visual perception and autonomy module at Plymouth university. It reports the technologies available in the Open-Computer-Vision (OpenCV) library and methodology to implement them using Python. For face detection, Haar-Cascades were used and for face recognition and Local binary pattern histograms were used. The methodology is described including flow charts for each stage of the system. Next, the results are shown including plots and screen-shots followed by a discussion of encountered challenges. The report is concluded with the authors' opinion on the project and possible applications.

INTRODUCTION

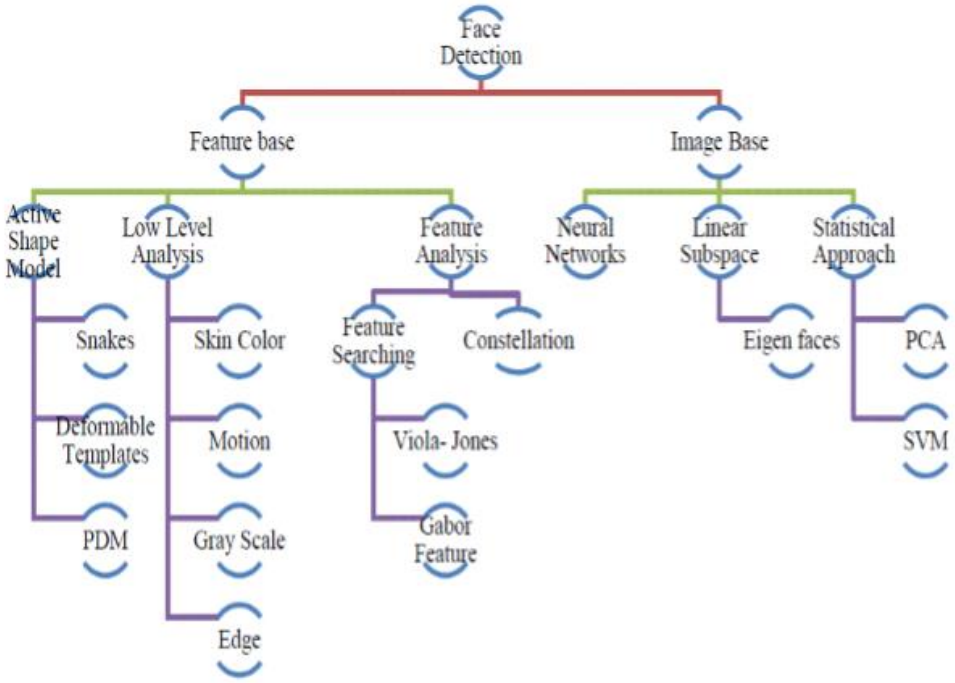
The following document is a report on the mini project for Robotic visual perception and autonomy. It involved building a system for face detection and face recognition using several classifiers available in the open computer vision library(OpenCV). Face recognition is a non-invasive identification system and faster than other systems since multiple faces can be analysed at the same time. The difference between face detection and identification is, face detection is to identify a face from an image and locate the face.

Face recognition is making the decision ”whose face is it ? ”, using an image database. In this project both are accomplished using different techniques and are described below. The report begins with a brief history of face recognition. This is followed by the explanation of Haar-cascades and Local binary pattern histogram (LBPH) algorithms. Next, the methodology and the results of the project are described. A discussion regarding the challenges and the resolutions are described. Finally, a conclusion is provided on the pros and cons of each algorithm and possible implementations.

LITERATURE SURVEY

Face detection is a computer technology that determines the location and size of human face in arbitrary (digital) image. The facial features are detected and any other objects like trees, buildings and bodies etc are ignored from the digital image. It can be regarded as a _specific‘ case of object-class detection, where the task is finding the location and sizes of all objects in an image that belong to a given class. Face detection, can be regarded as a more _general‘ case of face localization. In face localization, the task is to find the locations and sizes of a known number of faces (usually one). Basically there are two types of approaches to detect facial part in the given image i.e. feature base and image base approach.

Feature base approach tries to extract features of the image and match it against the knowledge of the face features. While image base approach tries to get best match between training and testing images.



THE HISTORY OF FACE RECOGNITION

Face recognition began as early as 1977 with the first automated system being introduced By Kanade using a feature vector of human faces [1]. In 1983, Sirovich and Kirby introduced the principal component analysis(PCA) for feature extraction [2]. Using PCA, Turk and Pentland Eigenface was developed in 1991 and is considered a major milestone in technology [3]. Local binary pattern analysis for texture recognition was introduced in 1994 and is improved upon

for facial recognition later by incorporating Histograms(LBPH) [4], [5]. In 1996 Fisherface was developed using Linear discriminant analysis (LDA) for dimensional reduction and can identify faces in different illumination conditions, which was an issue in Eigenface method [6]. Viola and Jones introduced a face detection technique using HAAR cascades and ADABoost [7]. In 2007, A face recognition technique was developed by Naruniec and Skarbek using Gabor Jets that are similar to mammalian eyes .In This project, HAAR cascades are used for face detection and Eigenface, Fisherface and LBPH are used for face recognition.

THEORY

1. FACE DETECTION USING HAAR-CASCADES

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 recognize signals with sudden transformations. An example is shown in figure 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 [10] for object detection as shown in figure 1.

To analyze 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 single classifier is not accurate enough.

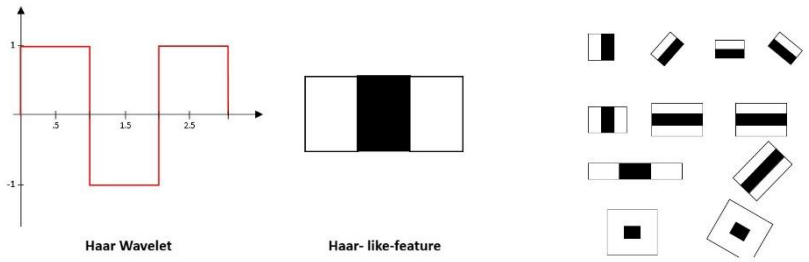


Figure 1: A Haar wavelet and resulting Haar-like features.

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 classifies were combined to create stronger classifiers. ADA boost is a machine learning algorithm that tests out several week 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.

Figure 2: Several Haar-like-features matched to the features of authors face.

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 [7] used a summed area table (an integral image) to compute the matches fast. First developed in 1984 [11], it became popular after 2001 when Viola Jones implemented Haar-cascades for face detection. Using an integral image enables matching features with a single pass over the image.

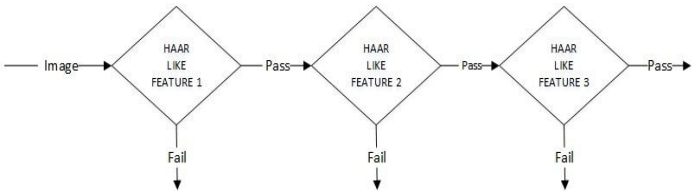


Figure 3: Haar-cascade flow chart

2. FACE RECOGNITION

The following sections describe the Local binary pattern histogram algorithm and how they are implemented in OpenCV.

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 [5]. 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 6. The value of intensity or luminosity of each neighbor is compared with the center 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

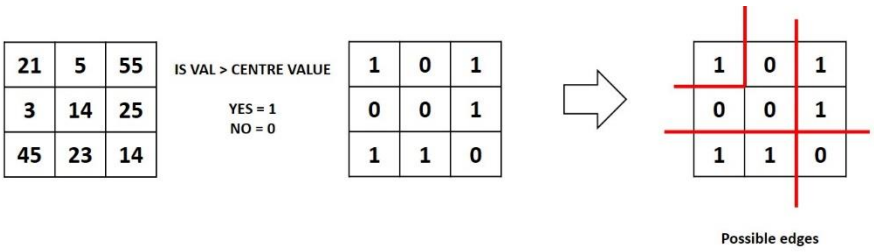


Figure 6: Local binary pattern histogram generating 8-bit number

Histograms are used in larger cells to find the frequency of occurrences of values making process faster. By analyzing 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.

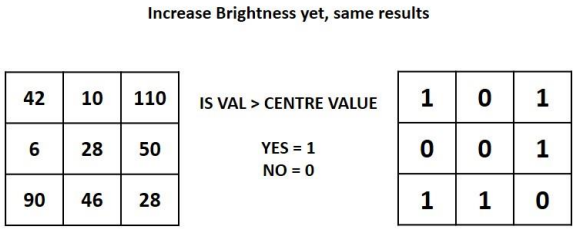


Figure 7: The results are same even if brightness is changed

3. OPEN CV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

4. IMAGE FILTERS

SHARPENING FILTER:

The unsharp filter is a straightforward sharpening operator which gets its name from the way that it improves edges by means of a system which subtracts an unsharp, or smoothed, rendition of a image from the first image. The unsharp filtering strategy is usually utilized in the photographic and printing ventures for crispening edges.

$$g(x,y) = f(x,y) - f_{smooth}(x,y)$$

where, g(x,y) = sharpened image
f(x,y) = original image
$f_{smooth}(x,y)$ = smoothed version of f(x,y)
unsharp mask used = [0 , -1 , 0 ; -1, 5,-1 ; 0, -1, -0]

MEAN FILTER:

It is a smoothening filter which is reduced to remove the intensity variation of the pixels in the image. It is also useful to remove noise present in the image. The kernel for Mean filter is:

9	9	9
9	9	9
9	9	9

This is used by pixel wrapping, pixel replication or by convolution with the image

DENOISE FILTER:

This filter removes any background noise that is present in the image and smoothens the image. Here non local means algorithm is used to remove noise. It uses the mean of the pixels of the image, checks the similarity of the pixels to the target pixel. It results in better removal of noise.

LAPLACIAN FILTER

Laplacian Operator is also a derivative operator which is used to find edges in an image. The major difference between Laplacian and other operators like Prewitt, Sobel, Robinson and Kirsch is that these all are first order derivative masks but Laplacian is a second order derivative mask. In this mask we have two further classifications one is Positive Laplacian Operator and other is Negative Laplacian Operator.

SOBEL OPERATOR

The sobel operator is very similar to Prewitt operator. It is also a derivate mask and is used for edge detection. Like Prewitt operator sobel operator is also used to detect two kinds of edges in an image:

- Vertical direction
- Horizontal direction

-1	0	+1
-2	0	+2
-1	0	+1

x filter

+1	+2	+1
0	0	0
-1	-2	-1

y filter

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 Python3.7 IDE and Spyder.

FACE DETECTION

First stage was creating a face detection system using Haar-cascades. Although, training is required for creating new Haar-cascades, OpenCV has a robust set of Haar-cascades that was used for the project. Using face-cascades alone caused random objects to be identified and eye cascades were incorporated to obtain stable face detection. The flowchart of the detection system can be seen in figure 8. Face and eye classifier objects are created using classifier class in OpenCV through the **cv2.CascadeClassifier()** and loading the respective XML files. A camera object is created using the **cv2.VideoCapture()** to capture images. By using the **CascadeClassifier.detectMultiScale()** object of various sizes are matched and location is returned. Using the location data, the face is cropped for further verification. Eye cascade is used to verify there are two eyes in the cropped face. If satisfied a marker is placed around the face to illustrate a face is detected in the location.

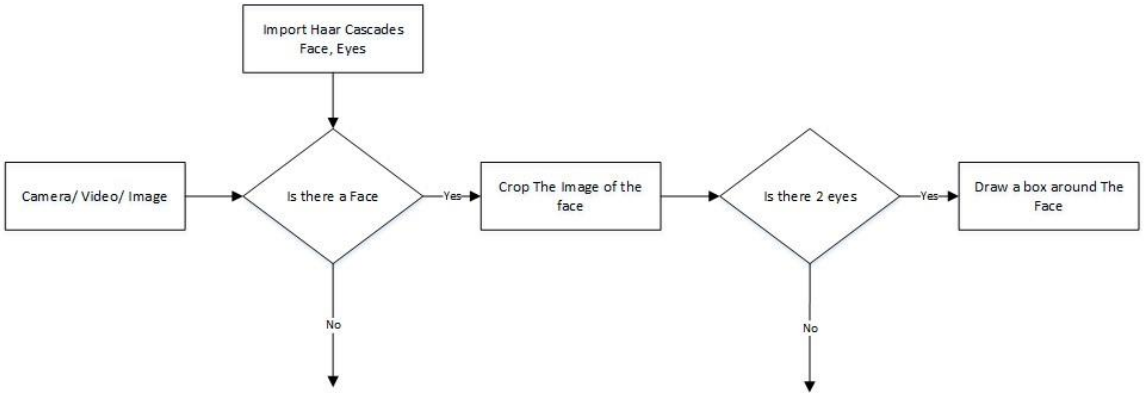


Figure 8: The Flow chart of the face detection application

FACE RECOGNITION PROCESS

For this project two algorithms are implemented independently. These are Haar Cascade 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.
4. Repeating the Same Process with enhancing the training images for better performance.

TRAINING THE CLASSIFIERS

OpenCV enables the creation of XML files to store features extracted from datasets using the **FaceRecognizer** 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 documentationmentions 80 can provide satisfactory reconstruction capabilities.
2. Takes in the threshold in recognising faces. If the distance to the likeliest Eigenface is above thisthreshold, the function will return a -1, that can be used state the face is unrecognisable **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 downthe 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 willreturn -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(NumpyImage, 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 a 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 10.

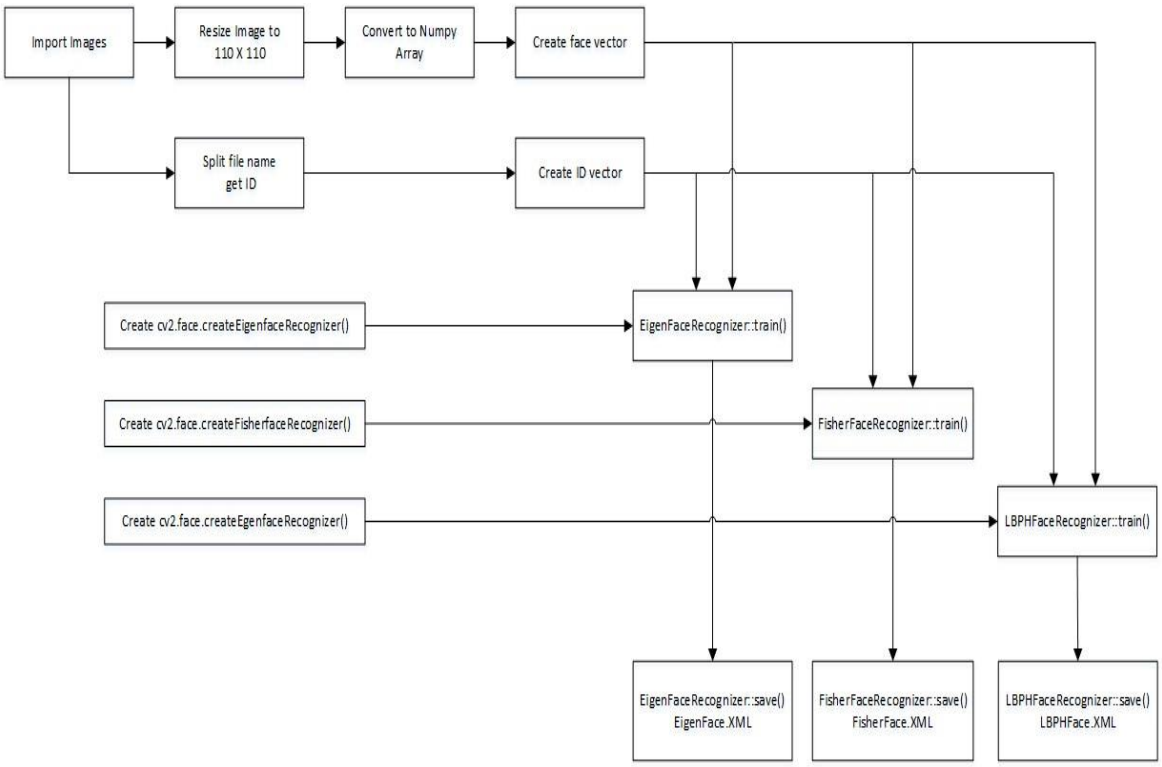


Figure 10: Flowchart of the training application

Face recognizer object is created using the desired parameters. Face detector is used to detect faces in the image, cropped and transferred to be recognized. 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 11.

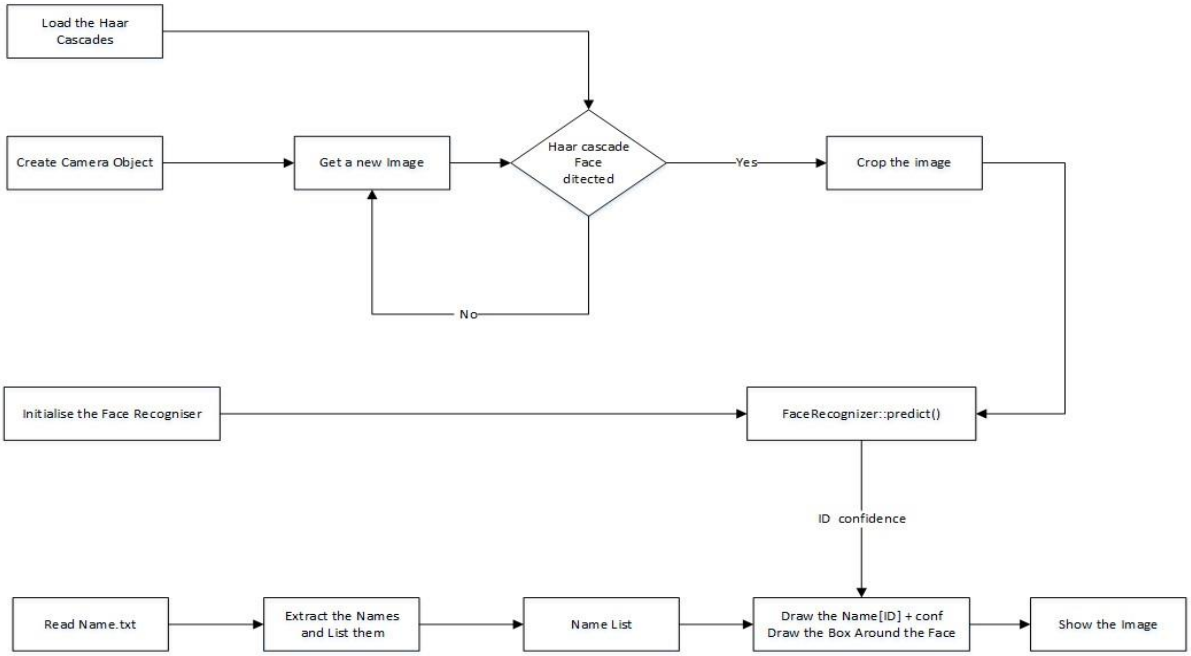


Figure 11: Flowchart of the face recognition application

ALGORITHM OF THE PROJECT

Step 1. The Webcam is opened from Laptop using Python code
Step 2. The video is showed on the screen at the same time it is recorded and stored in an .AVI file.
Step 3. The video which is been taken, is used for facial detection using haar cascade classifier.
Step 4. The video shows the detected face, eyes, nose, mouth and smile of the person provided there is enough backlight in the background.
Step 5. The video with detected face is also saved in an .AVI file for further processing.
Step 6. From both the videos, images(.JPG or .JPEG) are generated each for five frames per second.
Step 7. The images generated are stored in two folders separately.
Step 8. Image Processing is done for both grayscale and RGB Images.
Step 9. The images are used for image processing purposes

Step 10. Enhancement filters like brightness, contrast, denoise, sharpen, gamma transformation, histogram equalization are applied to create a perfect edited image.

Step 11. Laplacian operators, Sobel(horizontal and vertical) are applied to create the line and edge detected images of the original image.

Step 12. Further from the set of images generated from the video, they are used for facial recognition

Step 13. The facial recognition process is used with LBP and Haar cascade classifier for comparison purposes.

Step 14. The facial recognition process is repeated with a the of enhanced images for better results.

CODE SNIPPETS

Menu of the program

```
##.....................................................................MENU FOR PROGRAM .........
while True:
    print("MENU FOR FACIAL RECOGNITION WITH IMAGE ENHANCEMENTS\n")
    print("\n1. Convert the video to frames\n")
    print("\n2. Histogram Analysis of Image in BW Format\n")
    print("\n3. Equalised Histogram of Coloured Image\n")
    print("\n4. Appliying Filters in Spatial Domain\n")
    print("\n5. Enhance Training Pictures\n")
    print("\n6. Facial Recognition \n")
    print("\n7. Facial Recognition with Enhanced Images\n")
    x = int(input("\nEnter Your Choice (1-7): "))

    if x == 1:
        vid_to_frame(vidcap,vidcap2)

    elif x == 2:
        image_histogram()

    elif x == 3:
        BGR_Equalised_Histogram()

    elif x == 4:
        image_filters()

    elif x == 5:
        mypath='training-data/s3/'
        onlyfiles = [ f for f in listdir(mypath) if isfile(join(mypath,f)) ]
        images = np.empty(len(onlyfiles), dtype=object)
        for n in range(0, len(onlyfiles)):
          images[n] = cv2.imread( join(mypath,onlyfiles[n]) )
          x = mul_img(images[n],n)

    elif x == 6:
        fac_recog()

    elif x == 7:
        print('Face Recognition with Enchanced Images')
        fac_recog_edited()

    else:
        print('\nMenu Terminated')
        break
```

Face Detection with Haar Cascade

haar_cas_detect.py - D:\5TH_SEMESTER\CSE4019_IMAGE _PROCESSING\PROJECT\haar_cas_detect.py (3.5.4)

File Edit Format Run Options Window Help

```
import cv2
import numpy as np

def haar_cascade_face_detect():

    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
    eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
    nose_cascade = cv2.CascadeClassifier('Nariz.xml')
    mouth_cascade = cv2.CascadeClassifier('Mouth.xml')
    smile_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')

    cap = cv2.VideoCapture(0)
    fourcc = cv2.VideoWriter_fourcc(*'XVID')

    out = cv2.VideoWriter('output_normal.avi', fourcc, 20.0, (640,480))
    out2 = cv2.VideoWriter('output_detected.avi', fourcc, 20.0, (640,480))

    while True:

        ret, img = cap.read()
        frame = img
        out.write(frame)
        cv2.imshow('normal_capture',frame)
        #if ret!=0:
        gray = cv2. cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray,1.3,5)

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

            eyes = eye_cascade.detectMultiScale(roi_gray)
            for (ex,ey,ew,eh) in eyes:
                cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh),(0,255,0),2)

            nose_rects = nose_cascade.detectMultiScale(gray, 1.3, 5)
            for (x,y,w,h) in nose_rects:
                cv2.rectangle(img, (x,y), (x+w,y+h), (0,0,255), 3)

            mouth_rects = mouth_cascade.detectMultiScale(gray, 1.3, 5)
            for (x,y,w,h) in mouth_rects:
                cv2.rectangle(img, (x,y), (x+w,y+h), (0,128,0), 3)

                nose_rects = nose_cascade.detectMultiScale(gray, 1.3, 5)
                for (x,y,w,h) in nose_rects:
                    cv2.rectangle(img, (x,y), (x+w,y+h), (0,0,255), 3)

                mouth_rects = mouth_cascade.detectMultiScale(gray, 1.3, 5)
                for (x,y,w,h) in mouth_rects:
                    cv2.rectangle(img, (x,y), (x+w,y+h), (0,128,0), 3)

                smile = smile_cascade.detectMultiScale(roi_gray,1.7,22)
                for (sx,sy,sw,sh) in smile:
                    cv2.rectangle(roi_color, (sx,sy), (sx+sw,sy+sh),(0,255,255),1)

            cv2.imshow('face_detected',img)
            cv2.imwrite('detected_face_eyes_nose_mouth_smile.png',img)

            out2.write(img)

            k = cv2.waitKey(5) & 0xFF == ord('q')
            if k ==4:
                break
            else:
                pass

        cap.release()
        out.release()
        out2.release()
        cap.destroyAllWindows()


haar_cascade_face_detect()
```

Image Filters

```
def image_filters():

    img = cv2.imread('5.jpg')
    dst = cv2.fastNlMeansDenoisingColored(img,None,3,3,7,21)
    blur = cv2.blur(img,(5,5),50)
    kernel_sharpening = np.array([[0,-1,0],
                                  [-1, 5,-1],
                                  [0,-1,0]])
    sharpened = cv2.filter2D(img, -1, kernel_sharpening)

    #bright_img = cv2.add(img,30)
    contrast_img = cv2.multiply(img,1.1)

    enhanced_img = cv2.add(contrast_img,15)
    def gamma(img,g=1.00):
      invGamma = 1.0 / g
      table = np.array([((i / 255.0) ** invGamma) * 255
          for i in np.arange(0, 256)]).astype("uint8")
      return cv2.LUT(img, table)

    def bright_contr(img):
      brightness = 70
      contrast = 60
      img = np.int16(img)
      img0 = img * (contrast/127+1) - contrast + brightness
      img1 = np.clip(img0, 0, 255)
      img11 = np.uint8(img1)
      return img11

    img11 = bright_contr(img)

    img12 = gamma(img,1.2)


    fig = plt.figure(1)
    fig.subplots_adjust(hspace=0.5, wspace=0.5)
    plt.subplot(521),plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)),plt.title('Original Image')
    plt.subplot(522),plt.imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)),plt.title('Denoised Image')
    plt.subplot(523),plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)),plt.title('Original Image'),plt.xticks([]), plt.yticks([])
    plt.subplot(524),plt.imshow(cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)),plt.title('Blurred (Mean) Image'),plt.xticks([]), plt.yticks([])
    plt.subplot(525),plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)),plt.title('Original Image')
    plt.subplot(526),plt.imshow(cv2.cvtColor(sharpened, cv2.COLOR_BGR2RGB)),plt.title('Sharpened Image')
    plt.subplot(527),plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)),plt.title('Original Image')
    plt.subplot(528),plt.imshow(cv2.cvtColor(img11, cv2.COLOR_BGR2RGB)),plt.title('Bright  and contrasted')
```

Histogram, Colourised Histogram, Hostogram Equalisation

```
##.....................................................IMAGE HISTOGRAM.....................

def image_histogram():

    img = cv2.imread('2.jpg')
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    cv2.imwrite('Gray_messi.png',gray)
    gray_img = cv2.imread('Gray_messi.png',0)

    equ = cv2.equalizeHist(gray_img)
    res = np.hstack((gray_img,equ)) #stacking images side-by-side
    hist = cv2.calcHist([gray_img],[0],None,[256],[0,256])

    color = ('b','g','r')
    fig2 = plt.figure(1)
    for i,col in enumerate(color):
        histr = cv2.calcHist([img],[i],None,[256],[0,256])
        plt.subplot(122),plt.plot(histr,color = col),plt.xlim([0,256]),plt.title('Histogram of BGR Image')
        plt.subplot(121),plt.imshow(img),plt.title('BGR Image')
    plt.show()

    cv2.imwrite('res.png',res)

    fig = plt.figure(2)

    plt.subplot(221),plt.imshow(img),plt.title('Original Image')
    plt.subplot(222),plt.imshow(gray_img,'gray'),plt.title('Gray Image of Original')
    plt.subplot(223),plt.hist(gray_img.ravel(),256,[0,256]),plt.title('Histogram for gray scale picture')
    plt.subplot(224),plt.imshow(res,'gray'),plt.title('Equalised Histogram')
    plt.show()

    cv2.destroyAllWindows()
```

```
##.....................................................................COLOUR IMAGE EQUALISED HISTOGRAM .

def BGR_Equalised_Histogram():
    img = cv2.imread('4.jpg')

    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

    img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])

    img_output = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)

    cv2.imshow('Color input image', img)
    cv2.imshow('Histogram equalized', img_output)
    cv2.imwrite('Coloured_histo_2.jpg',img_output)
```

Laplacian Filters

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('Gray_messi.png',0)

laplacian = cv2.Laplacian(img,cv2.CV_64F)

sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

Batch Edit Images

```
##.....................................................................MULTIPLE IMAGES ENHANCEMENT .

def mul_img(img,n = 1):

    sharp = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
    sharp_img = cv2.filter2D(img, -1, sharp)
    gamma = 2.0
    invGamma= 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
        for i in np.arange(0, 256)]).astype("uint8")
    gamma_img =  cv2.LUT(sharp_img, table)
    cv2.imshow('sharp',gamma_img)

    cv2.imwrite("training-data-edited/w3/%d.png"%n,gamma_img)
    cv2.waitKey(200)
    cv2.destroyAllWindows()
```

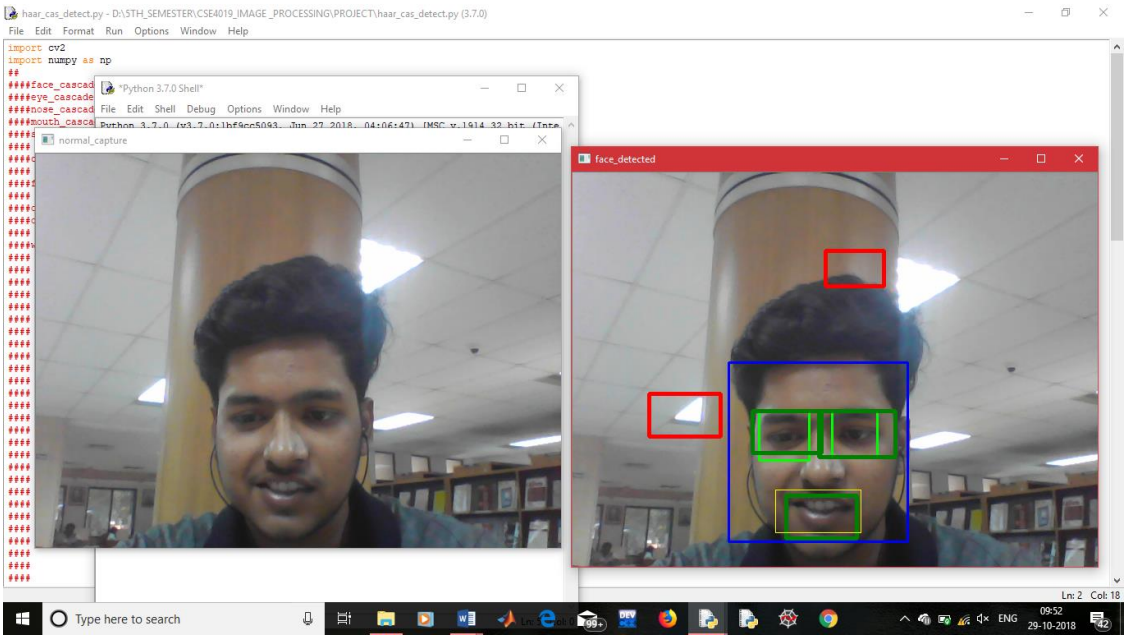
Facial Recognition

```
##.....................................................................FACIAL RECOGNITION...

def fac_recog():
    subjects = ["", "Wriddhirup Dutta", "Elvis Presley","Ramiz Raza"]

    def detect_face(img):

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        face_cascade = cv2.CascadeClassifier('opencv-files/lbpcascade_frontalface.xml')

        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5);

        if (len(faces) == 0):
            return None, None

        (x, y, w, h) = faces[0]

        return gray[y:y+w, x:x+h], faces[0]

    def prepare_training_data(data_folder_path):

        dirs = os.listdir(data_folder_path)

        faces = []

        labels = []

        for dir_name in dirs:

            if not dir_name.startswith("s"):
                continue;

            label = int(dir_name.replace("s", ""))

            subject_dir_path = data_folder_path + "/" + dir_name

            subject_images_names = os.listdir(subject_dir_path)

            for image_name in subject_images_names:

                if image_name.startswith("."):
                    continue;
```

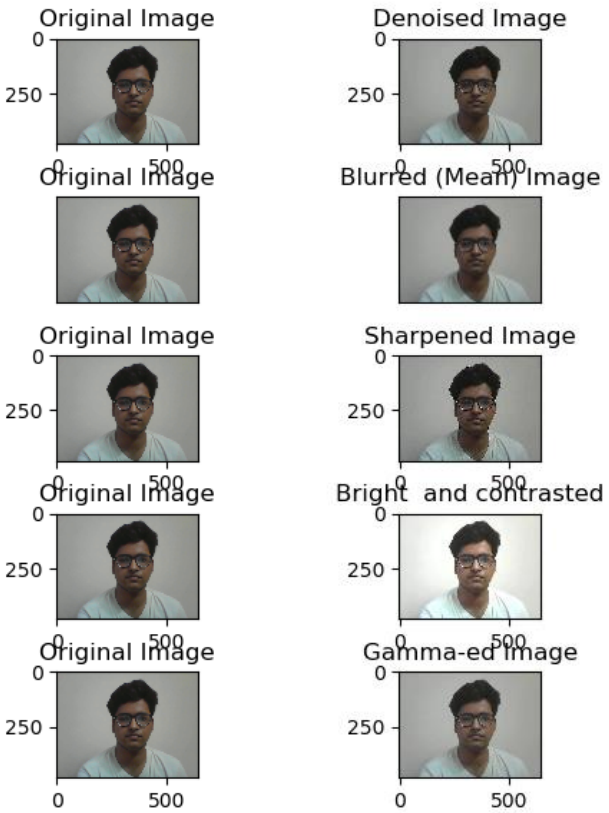
```
print("Preparing data...")
faces, labels = prepare_training_data("training-data")
print("Data prepared")

print("Total faces: ", len(faces))
print("Total labels: ", len(labels))

face_recognizer = cv2.face.LBPHFaceRecognizer_create()

face_recognizer.train(faces, np.array(labels))

def draw_rectangle(img, rect):
    (x, y, w, h) = rect
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

def draw_text(img, text, x, y):
    cv2.putText(img, text, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 2)

def predict(test_img):

    img = test_img.copy()

    face, rect = detect_face(img)

    label, confidence = face_recognizer.predict(face)

    label_text = subjects[label]

    draw_rectangle(img, rect)

    draw_text(img, label_text, rect[0], rect[1]-5)

    return img

print("Predicting images...")

test_img1 = cv2.imread("test-data/test1.jpg")
test_img2 = cv2.imread("test-data/test2.jpg")
test_img3 = cv2.imread("test-data/test3.jpg")

predicted_img1 = predict(test_img1)
predicted_img2 = predict(test_img2)
predicted_img3 = predict(test_img3)

print("Prediction complete")

cv2.imshow(subjects[1], cv2.resize(predicted_img1, (400, 500)))
cv2.imshow(subjects[2], cv2.resize(predicted_img2, (400, 500)))
cv2.imshow(subjects[3], cv2.resize(predicted_img3, (400, 500)))

cv2.imwrite("Predicted_Wriddhirup_1.jpg",predicted_img1)
cv2.imwrite("Predicted_Elvis_1.jpg",predicted_img2)
cv2.imwrite("Predicted_Ramiz_1.jpg",predicted_img3)
```

OUTCOMES

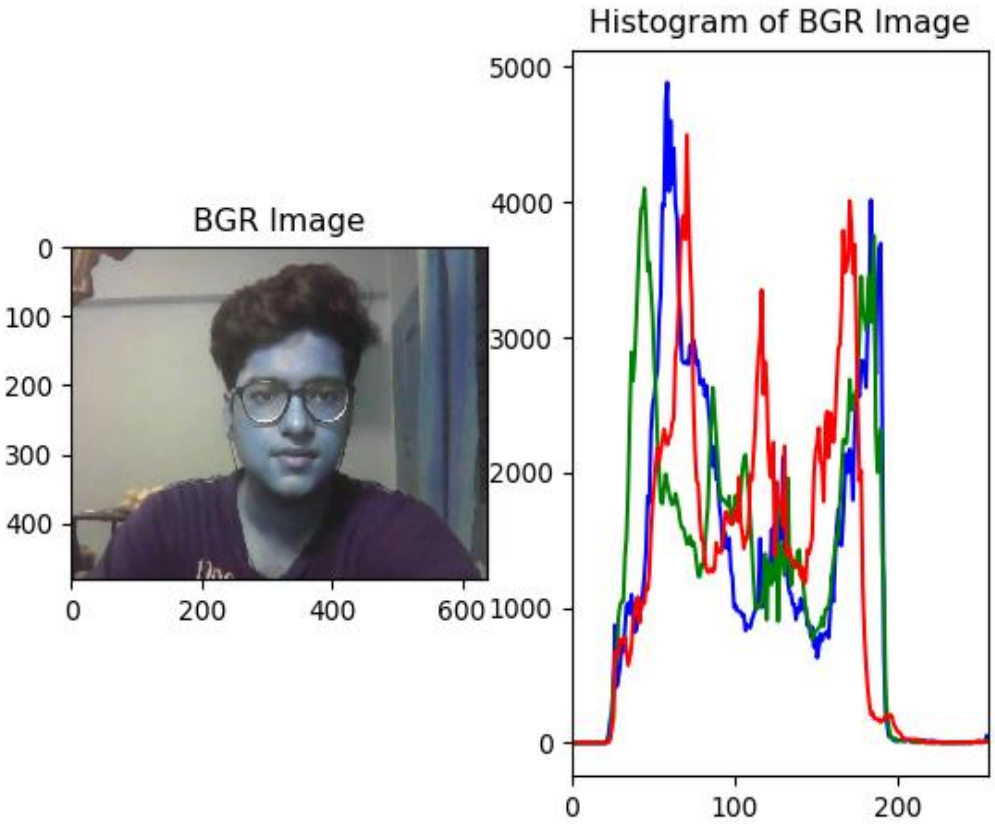
Face Detection

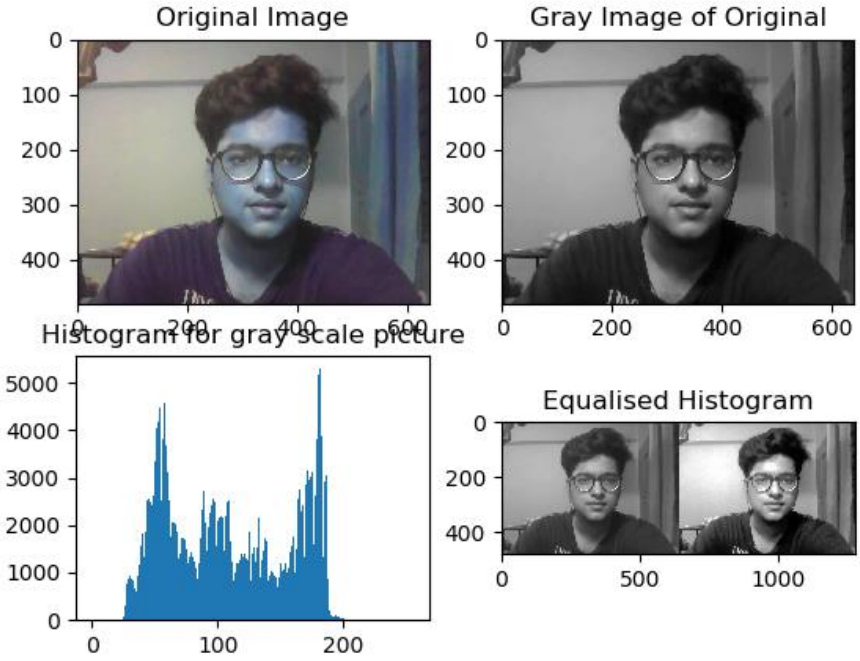


Edit Images

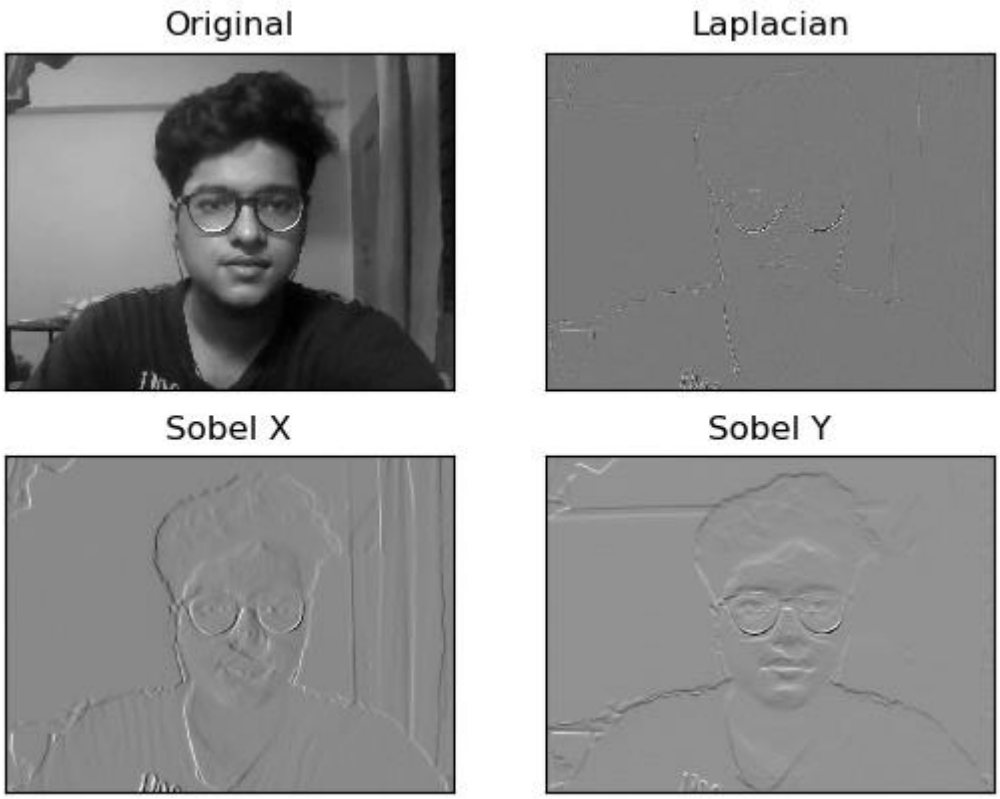


Histogram and Equalized Histogram

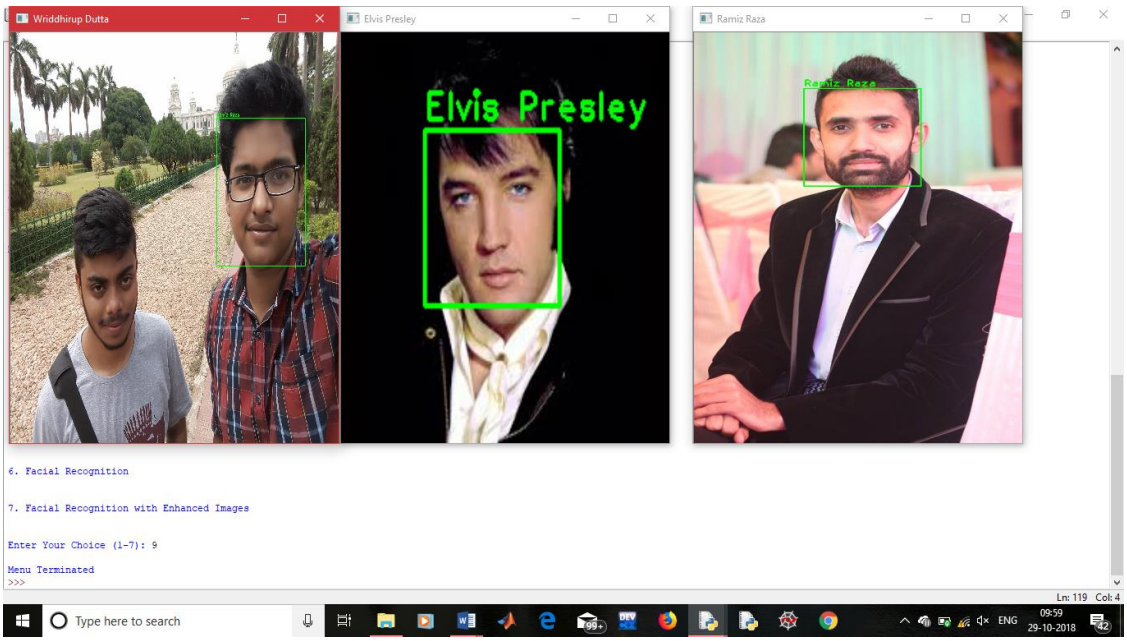




Laplacian and Sobel Operator Output:



Face Recognition Output



RESULT

The project has been successfully completed and the results have been generated successfully. Both Haar and LBP classifiers equally worked for our project.

- Haar Cascade classifier is more accurate and has low false detection rate. On the other hand it is slower, complex, less accurate to darker faces.
- LBP classifier is more faster, simple and takes less training time. It is robust to local illumination changes. But it is less accurate and has more false detection rate.

Thus, for more accurate results, Haar cascade classifier is preferred over LBP classifier. Facial recognition with LBP classifier is less time consuming.

With image enhancement, the recognition accuracy increases. Out of 10 test images with darker and lighter backgrounds and different facial expressions, it is able to recognize 9 images successfully. Thus, the algorithm is 90% efficient.

CONCLUSION

This project focuses on the application of image processing in real life scenarios with computer vision. We focussed on facial recognition and detection along with image enhancements. We have created an algorithm to edit multiple images from a folder with different filters at a time. It saves a lot of time rather than editing a single picture.

Here, two different classifier are used and the results for each cases are compared. It can be concluded that Haar cascade classifier is better than LBP classifier in most of the cases. The former produces more accurate results than the later and consumes more time. Haar classifier is able to detect more number of people in an image whereas LBP classifier is able to detect less faces. Both the classifiers work well in sufficient light. The classifiers are able to detect multiple faces from a video. Though LBP is computationally faster and simple, Haar classifier is preferred more when accurate results are needed. Although, LBP classifier works well even when the illumination is varied frequently.

REFERENCES

1. https://opencv.org/about.html
2. https://www.superdatascience.com
3. stackoverflow.com
4. github.com
5. Digital Image Processing (3rd Edition) by .Rafael C. Gonzalez, Richard E. Woods