A Minor Project Report

on

Face Recognition

Submitted in partial fulfilment of the requirement for the award of the degree

Of

BACHELORS OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

Submitted By

Piyush Kakkar

Enrolment No: 02996403114

Maharaja Agrasen Institute of Technology,

PSP area, Sector – 22, Rohini, New Delhi – 110085

(Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)

2014 – 2018

CERTIFICATE

This is to certify that the project report entitled "**Face Recognition**" being submitted by **Mr. Piyush Kakkar** in partial fulfilment for the award of the Degree of Bachelor of Technology in Information Technology to the Maharaja Agrasen Institute of Technology is a record of bonafied work carried out by him under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Dr. M.L. Sharma	Mr. Vibhor Sharma
(Head of the Department, IT)	(Assistant Professor, IT)

ACKNOWLEDGEMENT

I hereby take this opportunity to thank all those people whose knowledge and experience helped me bring this report in its present form. It would have been a tough task for me to complete this project without their help.

I would like to thank my HOD ***Dr. ML Sharma*** for his guidance and support. I express my sincere thanks and gratitude to my project guide ***Mr. Vibhor Sharma***, Assistant Professor, IT for providing me the opportunity to pursue my training at the office.

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

Piyush Kakkar

Enr. No: 02996403114

Information Technology Engineering

7 - IT - 7

ABSTRACT

Face detection and tracking has been an important and active research field. The goal of this project was to implement a real-time system to detect, track and recognize a human's face by their names and identifying multiple faces in real time. The software system uses face detection algorithms to accurately identify and recognize faces in real time. A pc webcam is used to record real time video data and the system automatically detects and identifies faces in real time application.

This system works as one of the futuristic of Artificial Intelligence and computer vision with user interface. This application can be used for wide variety of tasks including criminal identification, biometric security system, image and film processing, gaming, tagging purposes, image search and human-computer interaction.

This system uses Haar feature-based cascade classifiers approach for face detection. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Table of Contents

List of Figures

INTRODUCTION

1.1. Introduction

The face is our primary focus of attention in social life playing an important role in conveying identity and emotions. We can recognize a number of faces learned throughout our lifespan and identify faces at a glance even after years of separation. This skill is quite robust despite of large variations in visual stimulus due to changing condition, aging and distractions such as beard, glasses or changes in hairstyle.

Computational models of face recognition are interesting because they can contribute not only to theoretical knowledge but also to practical applications. Computers that detect and recognize faces could be applied to a wide variety of tasks including criminal identification, security system, image and film processing, identity verification, tagging purposes and human-computer interaction. Unfortunately, developing a computational model of face detection and recognition is quite difficult because faces are complex, multidimensional and meaningful visual stimuli.

Now a days, face detection is used in many places especially the websites hosting images like Picassa, Photobucket and Facebook. The automatically tagging feature adds a new dimension to sharing pictures among the people who are in the picture and also gives the idea to other people about who the person is in the image. In our project, we have studied and implemented a pretty simple but very effective face detection algorithm which takes human skin colour into account.

Our aim, which we believe we have reached, was to develop a method of face recognition that is fast, robust, reasonably simple and accurate with a relatively simple and easy to understand algorithms and techniques. The examples provided in this report are real-time and taken from our own surroundings.

Simple features such as color, motion, and texture are used for the face detection in early researches. However, these methods break down easily because of the complexity of the real world.

Face detection proposed by Viola and Jones is most popular among the face detection approaches based on statistic methods. This face detection is a variant of the AdaBoost algorithm which achieves rapid and robust face detection. They proposed a face detection framework based on the AdaBoost learning algorithm using Haar features. Therefore, this constitutes a bottleneck to the application of face detection in real time.

1.2. Face Detection Algorithm

The face detection algorithm proposed by Viola and Jones is used as the basis of our design. The face detection algorithm looks for specific Haar features of a human face. When one of these features is found, the algorithm allows the face candidate to pass to the next stage of detection. A face candidate is a rectangular section of the original image called a sub-window. Generally, these sub-windows have a fixed size (typically 24×24 pixels).

This sub-window is often scaled in order to obtain a variety of different size faces. The algorithm scans the entire image with this window and denotes each respective section a face candidate.

1.3. Integral Image

The simple rectangular features of an image are calculated using an intermediate representation of an image, called the integral image. The integral image is an array containing the sums of the pixels' intensity values located directly to the left of a pixel and directly above the pixel at location (x, y) inclusive. "Fig. 1" illustrates the integral image generation.

So if A[x, y] is the original image and AI[x, y] is the integral image then the integral image is computed as shown in equation 1 and illustrated in Fig. 2.

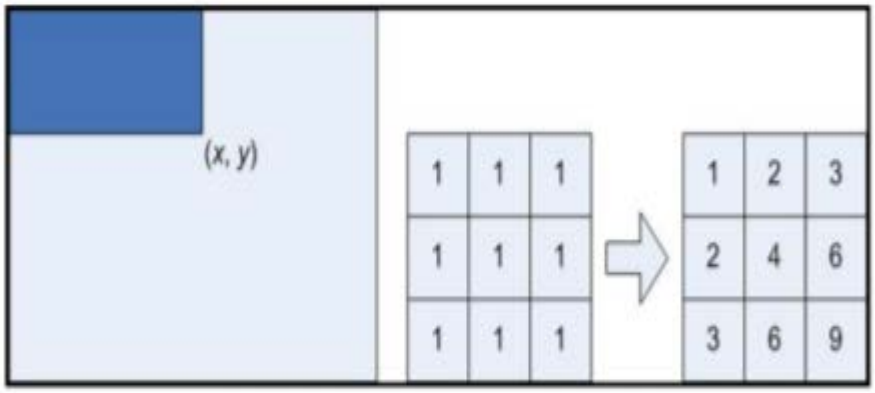

***Figure 1**: Integral image generation. The shaded region represents the sum of the pixels up to position (x, y) of the image. It shows a 3×3 image and its integral image representation.*

$$AI[x,y] = \sum_{x' \le x, y' \le y} A(x',y') \qquad (1)$$

The features rotated by forty-five degrees, like the line feature shown in Figure 4 2(e), as introduced by Lienhart and Maydt, require another intermediate representation called the rotated integral image or rotated sum auxiliary image. The rotated integral image is calculated by finding the sum of the pixels' intensity values that are located at a 45° angle to the left and above for the x value and below for the y value. So if A[x, y] is the original image and AR[x, y] is the rotated integral image then the integral image is computed as shown in equation 2 an illustrated in Figure 3.

$$AR[x,y] = \sum_{x' \le x, x' \le x - |y - y'|} A(x',y') \qquad (2)$$

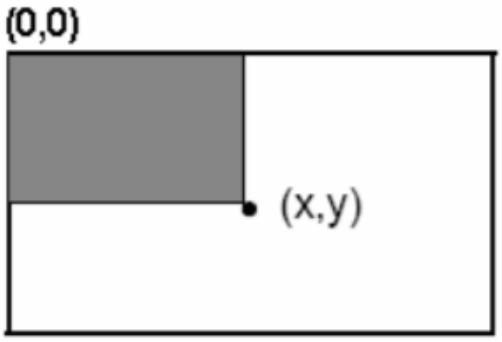

***Figure 3 :** Summed area of Integral Image*

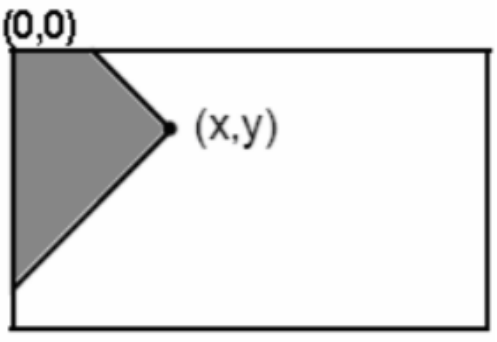

***Figure 3:** Summed area of rotated Integral Image*

It only takes two passes to compute both integral image arrays, one for each array. Using the appropriate integral image and taking the difference between six to eight array elements forming two or three connected rectangles, a feature of any scale can be computed. Thus, calculating a feature is extremely fast and efficient. It also means calculating features of various sizes requires the same effort as a feature of only two or three pixels. The detection of various sizes of the same object requires the same amount of effort and time as objects of similar sizes since scaling requires no additional effort.

1.4. Haar Features

Haar features are composed of either two or three rectangles. Face candidates are scanned and searched for Haar features of the current stage. The weight and size of each feature and the features themselves are generated using a machine learning algorithm from AdaBoost. The weights are constants generated by the learning algorithm. There are a variety of forms of features as seen below in “Fig. 5”.

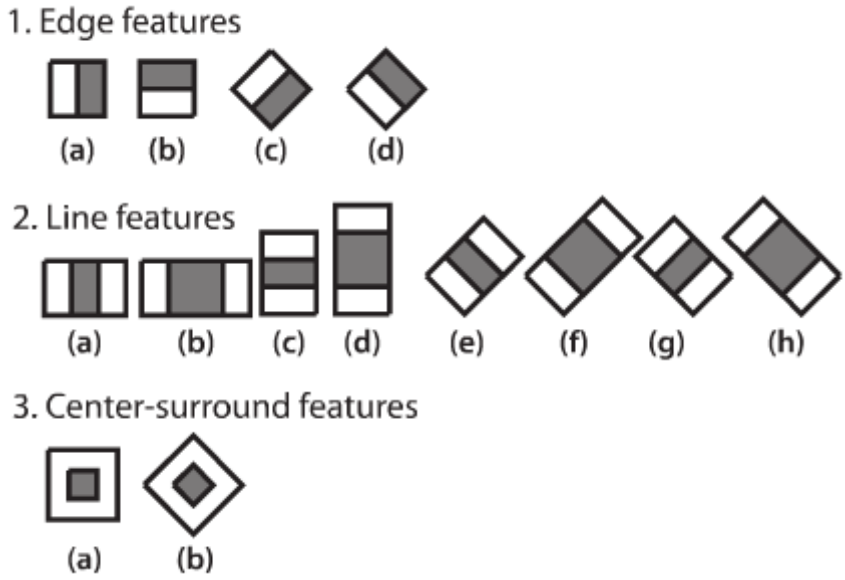

Figure 4: *Common Haar Features*

Figure 5: *Examples of Haar features. Areas of white and black regions are multiplied by their respective weights and then summed in order to get the Haar feature value.*

Each Haar feature has a value that is calculated by taking the area of each rectangle, multiplying each by their respective weights, and then summing the results. The area of each rectangle is easily found using the integral image. The coordinate of the any corner of a rectangle can be used to get the sum of all the pixels above and to the left of that location using the integral image. By using each corner of a rectangle, the area can be computed quickly as denoted by “Fig. 6”. Since L1 is subtracted off twice it must be added back on to get the correct area of the rectangle. The area of the rectangle R, denoted as the rectangle integral, can be computed as follows using the locations of the integral image: L4-L3-L2+L1.

1.5. Haar Feature Classifier

A Haar feature classifier uses the rectangle integral to calculate the value of a feature. The Haar feature classifier multiplies the weight of each rectangle by its area and the results are added together. Several Haar feature classifiers compose a stage. A stage comparator sums all the Haar feature classifier results in a stage and compares this summation with a stage threshold. The threshold is also a constant obtained from the AdaBoost algorithm. Each stage does not have a set number of Haar features. Depending on the parameters of the training data individual stages can have a varying number of Haar features. For example, Viola and Jones‟ data set used 2 features in the first stage and 10 in the second. All together they used a total of 38 stages and 6060 features. Our data set is based on the OpenCV data set which used 22 stages and 2135 features in total.

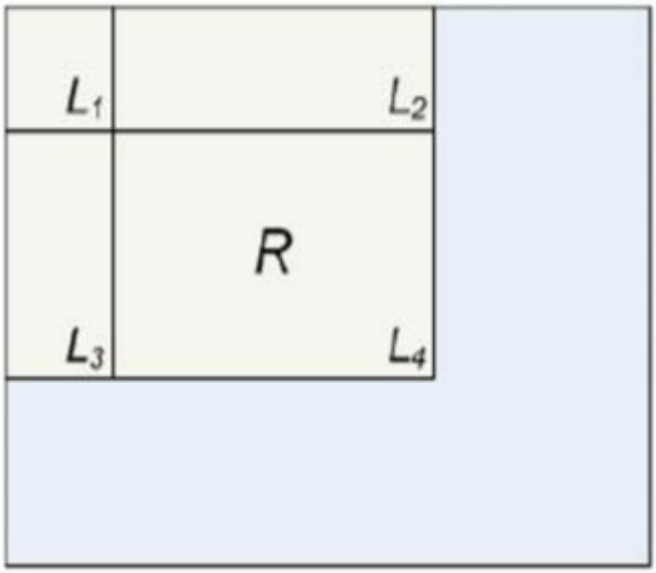

Figure 6: *Calculating the area of a rectangle R is done using the corner of the rectangle: L4-L3-L2+L1.*

1.6. Cascade

The Viola and Jones face detection algorithm eliminates face candidates quickly using a cascade of stages. The cascade eliminates candidates by making stricter requirements in each stage with later stages being much more difficult for a candidate to pass. Candidates exit the cascade if they pass all stages or fail any stage. A face is detected if a candidate passes all stages. This process is shown in “Fig. 7”.

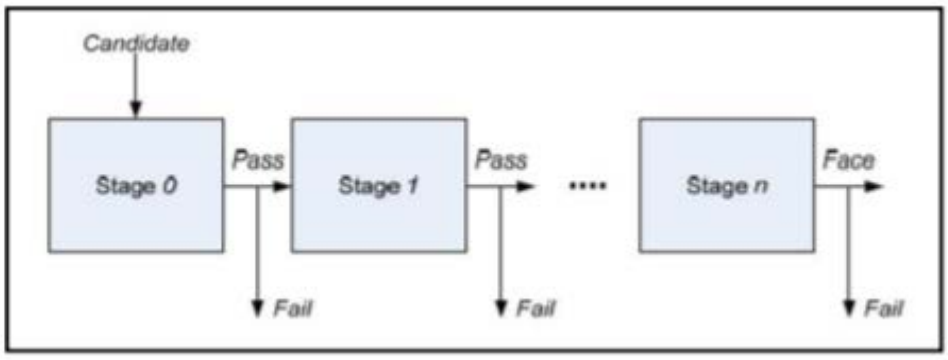

Figure 7: *Cascade of stages. Candidate must pass all stages in the cascade to be conclude*

1.7. Motivation

Face recognition has been a sought after problem of biometrics and it has a variety of applications in modern life. The problems of face recognition attracts researchers working in biometrics, patternrecognition field and computer vision . Several face recognition algorithms are also used in many different applications apart from biometrics , such as video compressions , indexings etc. They can also be used to classify multimedia content, to allow fast and efficient searching for material that is of interest to the user. An efficent face recognition system can be of great help in forensic sciences, identification for law enforcement, surveillance , authentication for banking and security system, and giving preferential access to authorised users i.e. access control for secured areas etc. The problem of face recognition has gained even more importance after the recent increase in the terrorism related incidents. Use of face recognition for authentication also reduces the need of remembering passwords and can provide a much greater security if face recognition is used in combination with other security measures for access control. The cost of the license for an efficient commercial Face recognition system ranges from 30,000 $ to 150,000 $ which shows the significant value of the problem. Though face recognition is considered to be a very crucial authentication system but even after two decades continuous research and evolution of many face recognition algorithms , a truely robust and efficient system that can produce good results in realtime and normal conditions is still not available. The Face Recognition Vendor Test (FRVT) that has been conducted by the National Institute of Standards and Technology (NIST), USA, has shown that the commercial face recognition systems do not perform well under the normal daily conditions. Some of the latest face recognition algorithm involving machine learning tools perform well but sadly the training period and processing time is large enough to limit its use in practical applications. Hence there is a continuous strife to propose an effective face recognition system with high accuracy and acceptable processing time.

SOFTWARE & HARDWARE REQUIREMENTS

2.1 Hardware Requirements

- Processor – Intel i5 or i7 5th generation (Minimum)
- RAM – 8GB
- Hard Disk – 500GB to 1TB
- WebCam

2.2 Software Requirements

- Python 3.x with following packages:- numpy+mkl
- OpenCV 3.3.0 + contrib
- Any recommended IDE for Python(used IDE is PyCharm)

LITERATURE SURVEY

During the last decade a number of promising face detection algorithms have been developed and published. Among these three stand out because they are often referred to when performance figures etc. are compared. This section briefly presents the outline and main points of each of these algorithms. Robust Real-Time Objection Detection, 2001 By Paul Viola and Michael J. Jones. This seems to be the first article where Viola-Jones presents the coherent set of ideas that constitute the fundamentals of their face detection algorithm. This algorithm only finds frontal upright faces, but is in 2003 presented in a variant that also detects profile and rotated views.

Neural Network-Based Face Detection, 1998 By Henry A. Rowley, Shumeet Baluja and Takeo Kanade. An image pyramid is calculated in order to detect faces at multiple scales. A fixed size sub window is moved through each image in the pyramid. The content of a sub window is corrected for non-uniform lightning and subjected to histogram equalization. The processed content is fed to several parallel neural networks that carry out the actual face detection. The outputs are combined using logical AND, thus reducing the amount of false detections. In its first form this algorithm also only detects frontal upright faces.

A Statistical Method for 3D Object Detection Applied to Faces and Cars, 2000 By Henry Schneider man and Takeo Kanade. The basic mechanics of this algorithm is also to calculate an image pyramid and scan a fixed size sub window through each layer of this pyramid. The content of the sub window is subjected to a wavelet analysis and histograms are made for the different wavelet coefficients. These coefficients are fed to differently train parallel that are sensitive to various orientations of the object. The orientation of the object is determined by the detector that yields the highest output. Opposed to the basic Viola Jones algorithm and the algorithm presented by Rowley et al. This algorithm also detects profile views. One of the fundamental problems of automated object detection is that the size and position of a given object within an image is unknown.

As two of the mentioned algorithms demonstrate the standard way to overcome this obstacle is to calculate an image pyramid and scan the detector through each image in the pyramid. While being relatively straightforward

to implement this process is rather time consuming, but in their paper Viola Jones presents a novel approach to this problem.

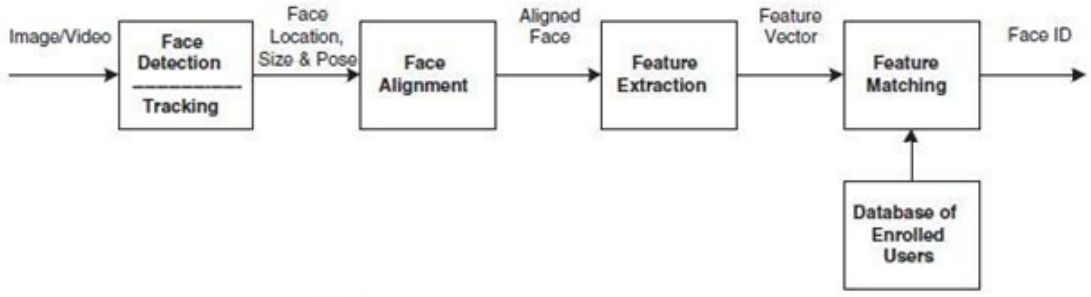

Figure 8: *Face Recognition Process Flow*

The project on face recognition had helped the author to have a detailed survey of a number of face recognition algorithms along with their advantages and limitations. Some of the important methods studied will be described in this section. Face recognition systems architecture broadly consists of the three following tasks:

- Acquisition (Detection, Tracking of face-like images)
- Feature extraction (Segmentation, alignment & normalization of the face image)
- Recognition

2.1 Face Detection Approaches

Some of the main face detection methods are discussed here.

- **Knowledge based methods** are developed on the rules derived from the researcher's knowledge of human faces. Problem in this approach is the difficulty in translating human knowledge into well-defined rules.
- **Featured-based methods:** Invariant features of faces are used for detecting texture, skin colour. But features from such algorithm can be severely corrupted due to illumination, noise and occlusion.
- **Template matching:** Input image is compared with predefined face template. But the performance here suffers due to variations in scale, pose and shape.

- **Appearance-based method:** In template matching methods, the templates are predefined by experts. Whereas, the templates in appearance based methods are learned from examples in images. Statistical analysis and machine learning techniques can be used to find the relevant characteristics of face and non-face images.

2.2 Face Recognition Approaches

LFA method of recognition analyses the face in terms of local features, e.g., eyes, nose, etc. referred to as LFA kernels. LFA technique offers better robustness against local variations on the facial image in carrying out a match, but does not account for global facial attributes. Neural Network are based on learning of the faces in an example set by the machine in the 'training phase and carrying out recognition in the 'generalization phase. But in order to succeed in a practical set-up, the examples should sufficiently large in number to account for variations in real life situations. Model Matching methods of face recognition (like Hidden Markov Model (HMM)) train a model for every person during model learning and choose the best matching model, given a query image. Here also a big realistic representative model is necessary for good results. A recognition system based on sparse representation computed by l^1 -minimization works with the basic idea of casting the recognition as a sparse representation problem. The main concern in this approach is the presence of sufficiently large number of features and correct computation of sparse representation. It is a robust and scalable algorithm for face recognition based on linear or convex programming.

SOFTWARE REQUIREMENT ANALYSIS

4.1 Problem Statement

Considering of an image representing a frame taken from video stream, automatic face recognition is a particularly complex task that involves detection and location of faces in a cluttered background followed by normalization and recognition. The human face is a very challenging pattern to detect and recognize, because while its anatomy is rigid enough so that all faces have the same structure, at the same time there are a lot of environmental and personal factors affecting facial appearance.

The main problem of face recognition is large variability of the recorded images due to pose, illumination conditions, facial expressions, use of cosmetics, different hairstyle, presence of glasses, beard, etc. Images of the same individual taken at different times, may sometimes exhibit more variability due to the aforementioned factors (intrapersonal variability), than images of different individuals due to gender, race, age and individual variations (extra personal variability).

One way of coping with intrapersonal variations is including in the training set images with such variations. And while this is a good practice for variations such as facial expressions, use of cosmetics and presence of glasses or beard, it may not be successful in case of illumination or pose variations.

4.2 Modules and their Functionalities

4.2.1 Training Module

Training module will perform the following functions

I. It will consist of a sub module that will take care of loading the entire dataset of images used for training the model.
II. This sub module will also take care of converting these images into array representation in Python.
III. Then the control will be passed on further to a sub module that will convert this array representation into a NumPy vector space.

IV. A NumPy vector space is more suitable for vision based operations than a normal list based approach.
V. For example, if a dataset has 250 coloured images each of 100x100 dimensionality, the shape of numpy array would be (250,100,100,3) where 250 is the batch size, 100,100 is the dimensionality and 3 is the number of channels for the RGB mode.
VI. After normalization of the dataset and reshaping of the labels, it will call another sub module which will generate various layers of the Convolutional Neural Network.
VII. Then another sub module for the compilation and fitting of the model according to the dataset will be called.
VIII. This saved model can now be used for classification in other modules.

4.2.2 Capture and Manipulation Module

Capture and manipulation module will take care of:

I. Capturing the image feed of the webcam for processing.
II. Processing this image for identifying and converting it into grayscale image.
III. Grayscale images are stored in database used for training of model.
IV. Grayscale images are compared with images obtained from video stream and face is recognized.

4.2.3 Segmentation Module

Segmentation module will perform the following functions

I. The Segmentation Module will make use of Haar classification.
II. Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Now all possible sizes and locations of each kernel is used to calculate plenty of features. For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the

integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels.

III. Haar Classification will allow segmentation between face and its surroundings.

4.2.4 Conclusion Module

The function of the conclusion module is to call all the other primary modules and display the result at the end.

SOFTWARE DESIGN

5.1 Sequence Diagram

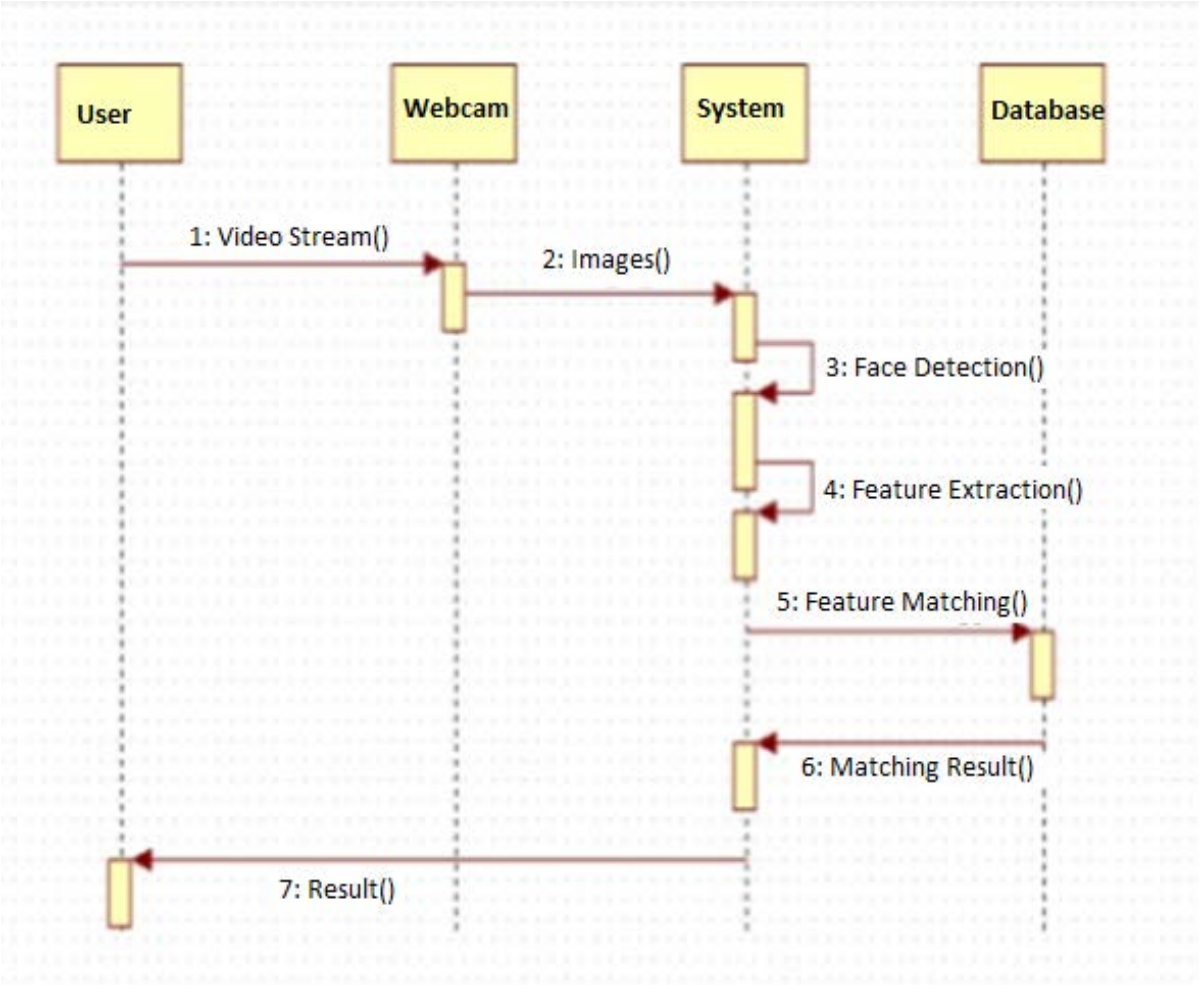

Figure 9: *Sequence Diagram*

- First a training script is made to run the user may bring his face in front of the camera and the system allows the user to get a snapshot of his face and enter his/her name.
- The system records the images and maintains it in its system. Similarly, multiple users may record their data in the system.

- Now when face recognition script is made to run it starts capturing real time video data again. Haar Features are extracted from these images.
- As a previously registered user appears in the video, the system matches his face to previously stored records.
- The system then recognizes the user and displays the name of the user appearing in the video.

5.2 Use Case Diagram

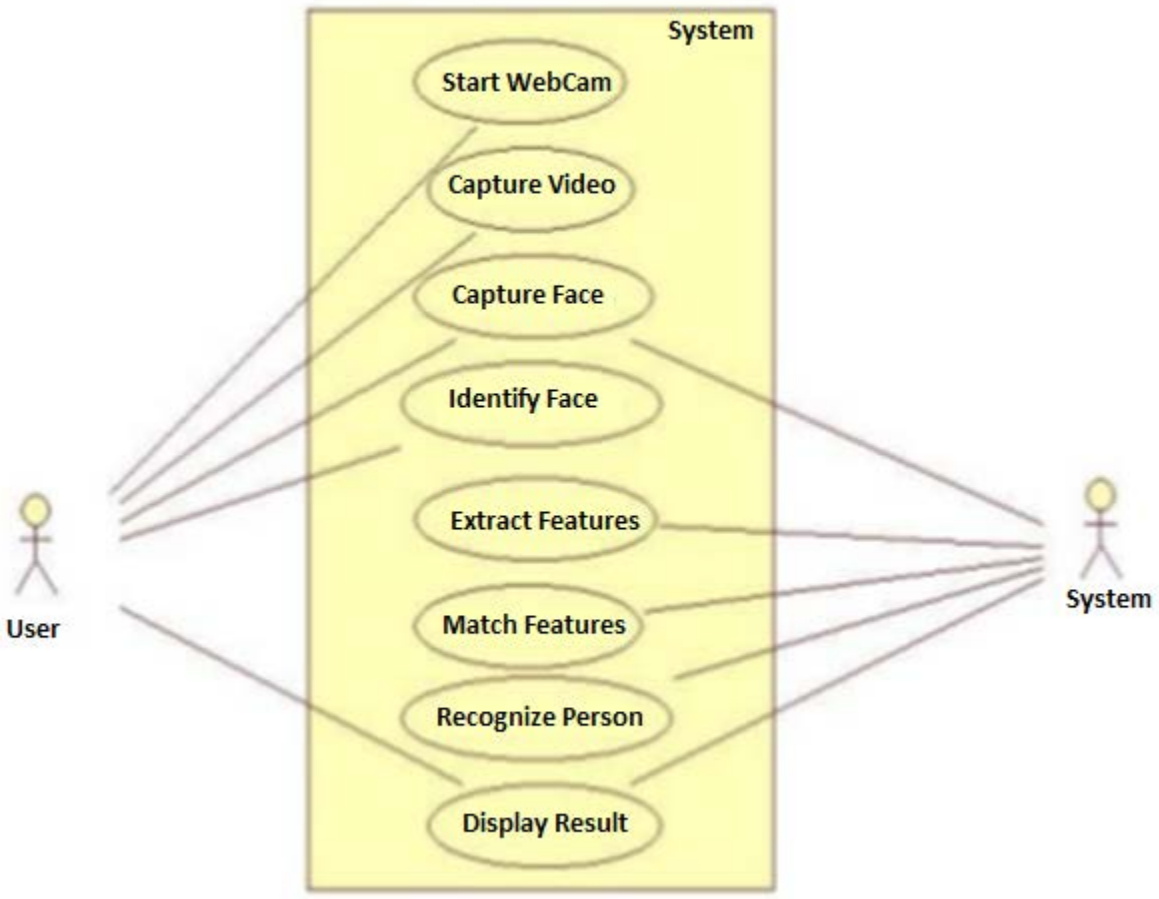

Figure 10: *Use Case Diagram*

5.3 Flowchart

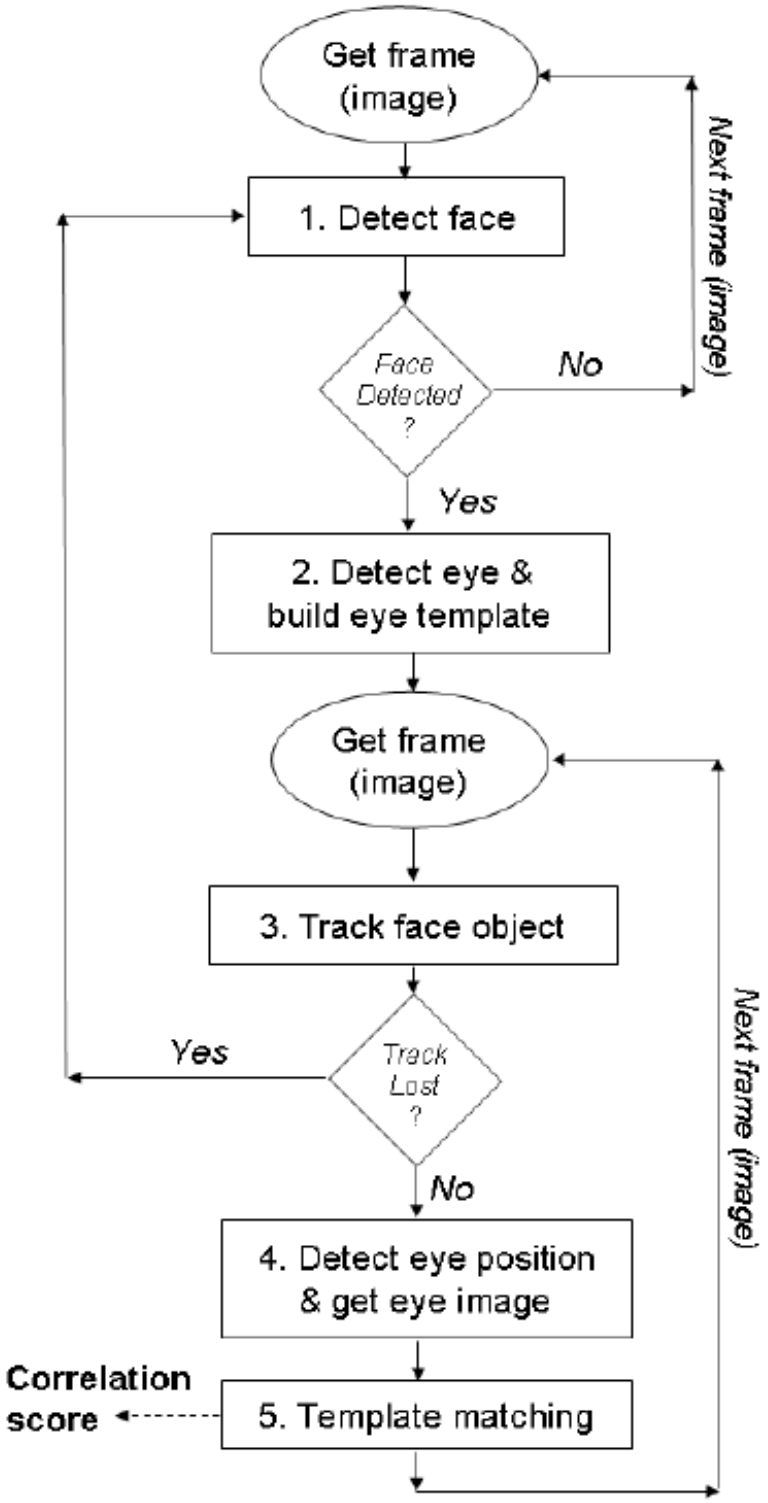

Figure 11: *Flow Chart*

Code Templates

6.1 facerec.py

6.2.1 Getting started with OpenCV

The first part of the program is getting the input from your webcam.

The Python program simply loops capturing and showing an image. To exit the program, either press Exit or Ctrl-C . If you have multiple cameras, you might have to change the '0' in line 4 to '1' or '2' (and so on).

At this point, the program isn't doing much. It's just receiving a video stream from the webcam and displaying it. The next step is getting it to detect the faces we want to identify.

For this part, you are going to need a cascade file named as haarcascade_frontalface_default.xml for detecting faces.

```
# facerec.py
import cv2

# Use camera 0 (you might need to change this if you have
multiple cameras)
webcam = cv2.VideoCapture(0)

while True:

    # Loop until the camera is working
    rval = False
    while(not rval):
        # Put the image from the webcam into 'frame'
        (rval, frame) = webcam.read()
        if(not rval):
            print("Failed to open webcam. Trying again..")

    # Flip the image (optional)
    frame=cv2.flip(frame,1,0)

    # Show the image in a window with title "OpenCV"
    cv2.imshow("OpenCV", frame)

    # Wait for
    key = cv2.waitKey(10)
    if key == 27: #The Esc Key
        break
```

6.2.1 Detecting Faces

```
# facerec.py
import cv2
size = 1
webcam = cv2.VideoCapture(0) #Use camera 0

# We load the xml file
classifier =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml
')

while True:

    # Loop until the camera is working
    rval = False

    while(not rval):
        # Put the image from the webcam into 'frame'
        (rval, frame) = webcam.read()
        if(not rval):
            print("Failed to open webcam. Trying again..")

    frame=cv2.flip(frame,1,0) #Flip to act as a mirror

    # Resize the image to speed up detection
    mini = cv2.resize(frame, (int(frame.shape[1] / size),
                              int(frame.shape[0] / size)))

    # We let OpenCV do it's thing
    faces = classifier.detectMultiScale(mini)

    # Draw rectangles around each face
    for f in faces:
        (x, y, w, h) = [v * size for v in f]
    #Scale the shapesize backup
        cv2.rectangle(frame, (x, y), (x + w, y + h),
                                  (0,255,0), thicknwess=4)

    # Show the image and lookout for the Escape key
    cv2.imshow('OpenCV', frame)
    key = cv2.waitKey(10)
    if key == 27: #The Esc key
        break
```

6.2.1 Creating Fisher Face Recognizer

```
# Part 1: Create fisherRecognizer
print('Training...')

# Create a list of images and a list of corresponding
names
```

```
(images, lables, names, id) = ([], [], {}, 0)

# Get the folders containing the training data
for (subdirs, dirs, files) in os.walk(fn_dir):

    # Loop through each folder named after the subject in
     the photos
    for subdir in dirs:
        names[id] = subdir
        subjectpath = os.path.join(fn_dir, subdir)

        # Loop through each photo in the folder
        for filename in os.listdir(subjectpath):
            # Skip non-image formates
            f_name, f_extension=os.path.splitext(filename)
            if(f_extension.lower() not in
                ['.png', '.jpg', '.jpeg', '.gif', '.pgm']):
                print("Skipping "+filename+", wrong file
                                                    type")
                continue
            path = subjectpath + '/' + filename
            lable = id

            # Add to training data
            images.append(cv2.imread(path, 0))
            lables.append(int(lable))
        id += 1
(im_width, im_height) = (112, 92)

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

# OpenCV trains a model from the images
# NOTE FOR OpenCV2: remove '.face'
model = cv2.face.FisherFaceRecognizer_create()
model.train(images, lables)

print('Training Successful. Detecting Faces')
```

6.2.1 Recognizing Face of a Person

```
# Part 2: Use fisherRecognizer on camera stream
haar_cascade = cv2.CascadeClassifier(fn_haar)
webcam = cv2.VideoCapture(0)
while True:

    # Loop until the camera is working
    rval = False
    while(not rval):
        # Put the image from the webcam into 'frame'
        (rval, frame) = webcam.read()
```

```
        if(not rval):
            print("Failed to open webcam. Trying again..")

    # Flip the image (optional)
    frame=cv2.flip(frame,1,0)

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

    # Resize to speed up detection (optinal, change size
above)
    mini = cv2.resize(gray, (int(gray.shape[1] / size),
                             int(gray.shape[0] / size)))

    # Detect faces and loop through each one
    faces = haar_cascade.detectMultiScale(mini)
    for i in range(len(faces)):
        face_i = faces[i]

        # Coordinates of face after scaling back by `size`
        (x, y, w, h) = [v * size for v in face_i]
        face = gray[y:y + h, x:x + w]
        face_resize = cv2.resize(face, (im_width,
                                        im_height))

        # Try to recognize the face
        (prediction, confidence) =
                              model.predict(face_resize)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0,
                                        0, 255), 3)

        # Write the name of recognized face
        # print(prediction, confidence)
        if confidence>500:
            cv2.putText(frame,'Unknown',(x - 10, y - 10),
            cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 255), 2)
        else:
            cv2.putText(frame,'%s - %d' %
               (names[prediction], confidence),
               (x - 10, y - 10), cv2.FONT_HERSHEY_PLAIN,
               1.5, (0, 255, 255), 2)


    # Show the image and check for ESC being pressed
    cv2.imshow('OpenCV', frame)
    key = cv2.waitKey(10)
    if key == 27:
       break
```

6.2 train.py

This file will add to the training set using images from the webcam. Make sure there is only one face in the shot.

```
# train.py
import cv2, sys, numpy, os

size = 1
try:
    fn_name = sys.argv[1]
except:
    print("You must provide a name")
    sys.exit(0)

path = os.path.join('face_samples', fn_name)

if not os.path.isdir(path):
    os.mkdir(path)
(im_width, im_height) = (112, 92)

haar_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml
')

webcam = cv2.VideoCapture(0)

# Generate name for image file
pin=sorted([int(n[:n.find('.')]) for n in os.listdir(path)
     if n[0]!='.' ]+[0])[-1] + 1

# Beginning message
print("\nThe program will save 20 samples. Move your head
around to increase while it runs.\n")

# The program loops until it has 20 images of the face.
count = 0
pause = 0
count_max = 20

while count < count_max:

    # Loop until the camera is working
    rval = False
    while(not rval):
        # Put the image from the webcam into 'frame'
        (rval, frame) = webcam.read()
        if(not rval):
            print("Failed to open webcam. Trying again..")

    # Get image size
    height, width, channels = frame.shape
```

```
# Flip frame
frame = cv2.flip(frame, 1, 0)

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

# Scale down for speed
mini = cv2.resize(gray, (int(gray.shape[1] / size),
                         int(gray.shape[0] / size)))

# Detect faces
faces = haar_cascade.detectMultiScale(mini)

# We only consider largest face
faces = sorted(faces, key=lambda x: x[3])
#sort based on fourth value of quad

if faces:
    face_i = faces[0]
    (x, y, w, h) = [v * size for v in face_i]

    face = gray[y:y + h, x:x + w]
    face_resize = cv2.resize(face, (im_width,
                                    im_height))

    # Draw rectangle and write name
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0,
                                    255, 0), 3)

    # Remove false positives
    if(w * 6 < width or h * 6 < height):
        print("Face too small")
    else:
        # To create diversity, only save every fifth
        detected image
        if(pause == 0):
            print("Saving training sample " +
                str(count+1) + "/"+str(count_max))

            # Save image file
            cv2.imwrite('%s/%s.png' % (path, pin),
                                    face_resize)
            pin += 1
            count += 1
            pause = 1

if(pause > 0):
    pause = (pause + 1) % 5
cv2.imshow('OpenCV', frame)
key = cv2.waitKey(10)
if key == 27:
    break
```

Save this as train.py and run it using $ python path/to/file/train.py Name where path/to/file/ is the path to the folder containing the file, and Name is the name of the subject. train.py will use the face *detection* code from above. It will store 20 images of the face it detects and stores it in face_samples under the name you gave it.

Output Screens

1 Running face recognition script when the model is not trained for a particular person. At that time the face in the screen will be considered as unknown face.

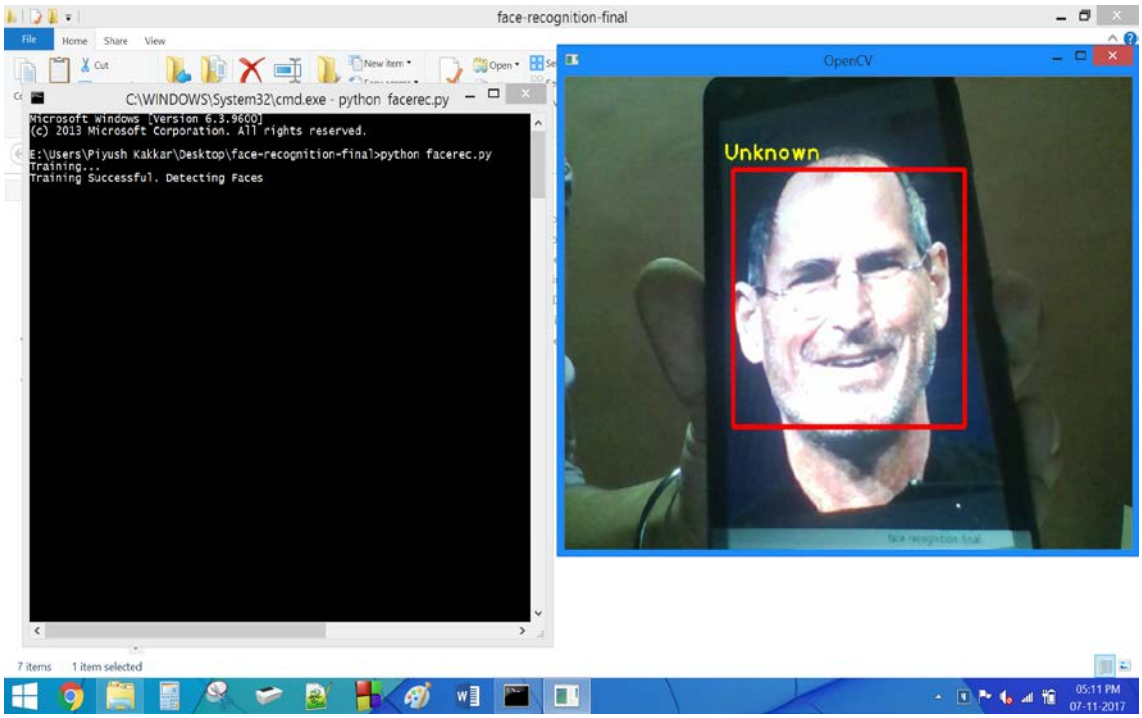

Figure 12: *Output: Unknown Face*

2 Running training script to train our model to recognize a particular person. Training script will capture 20 images of face. These images are converted to grayscale and are scaled to region where face is present.

To create diversity we only save every fifth detected image.

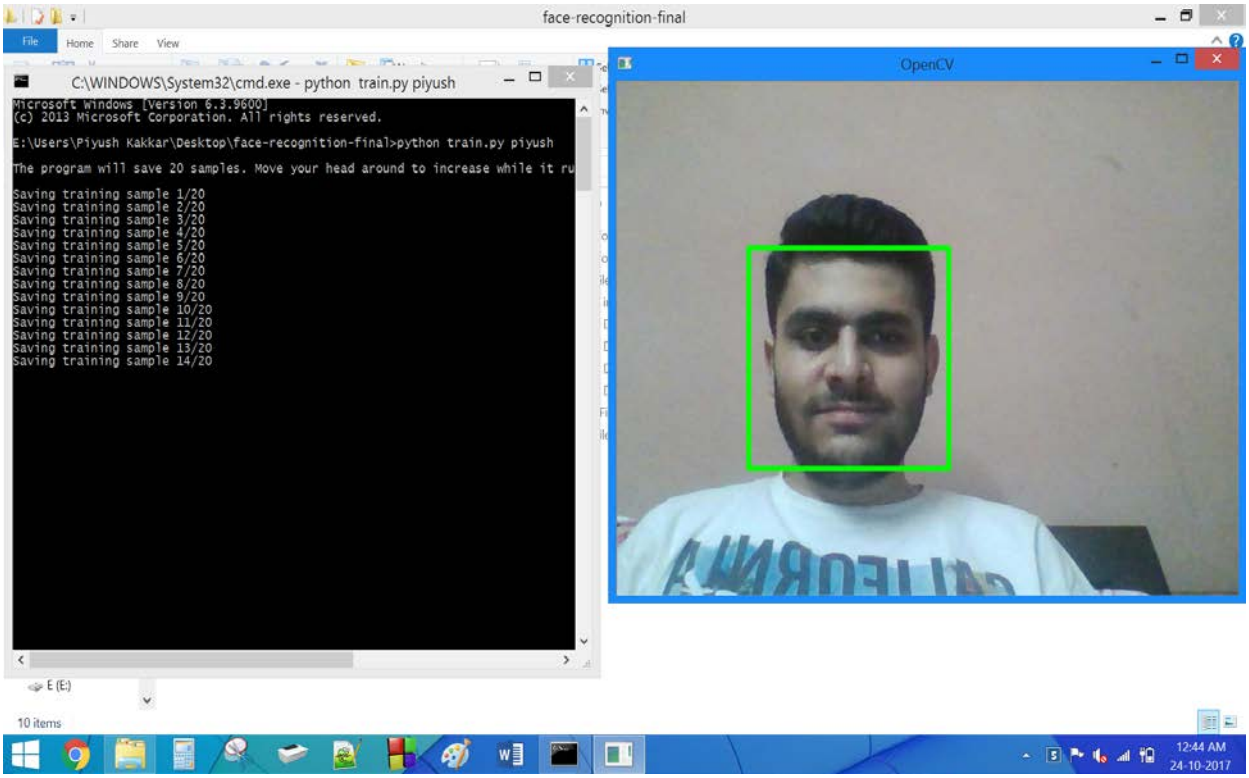

Figure 13: *Output: Training*

3 Running face recognition script when the model is trained for a particular person. At that time the face in the screen will be recognized and name is displayed along with the confidence with which the face of the person is detected.

Lower the confidence value better is the face recognized.

The system will also recognize the person when the face is at different angles.

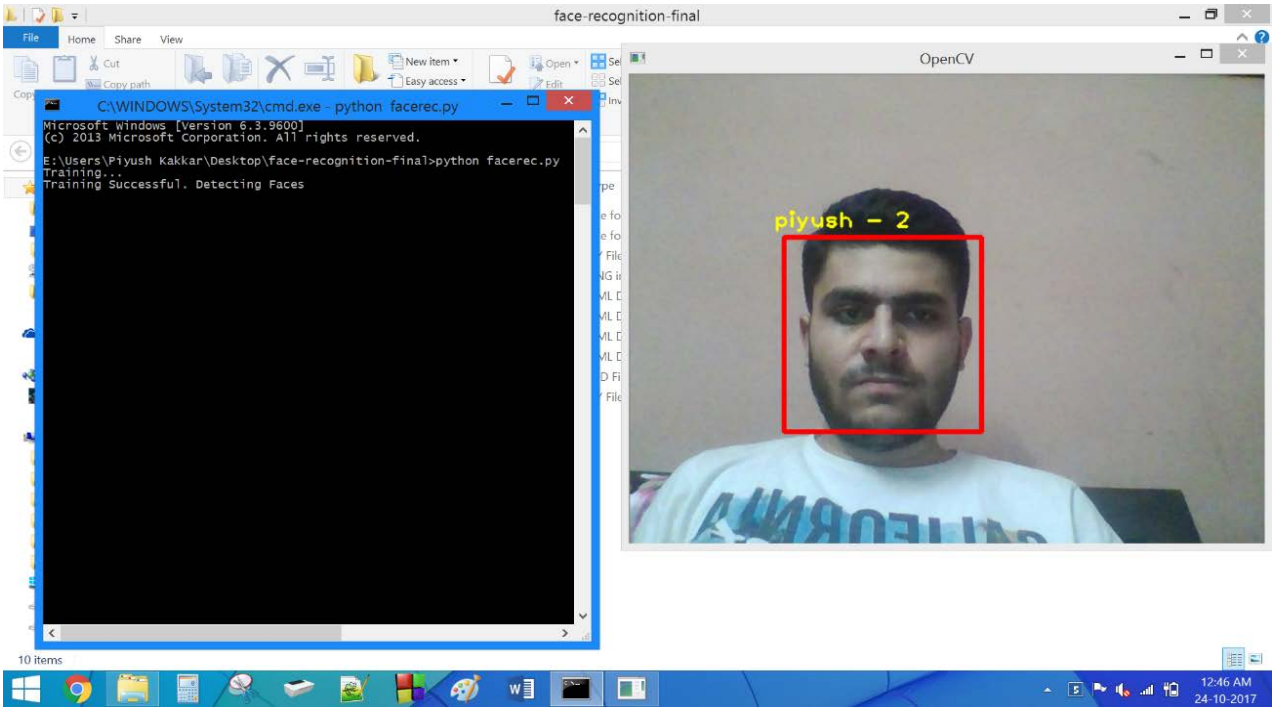

Figure 14: *Output: Recognized Face with Confidence Value*

4 Our system is also capable detecting multiple face from the video stream simultaneously.

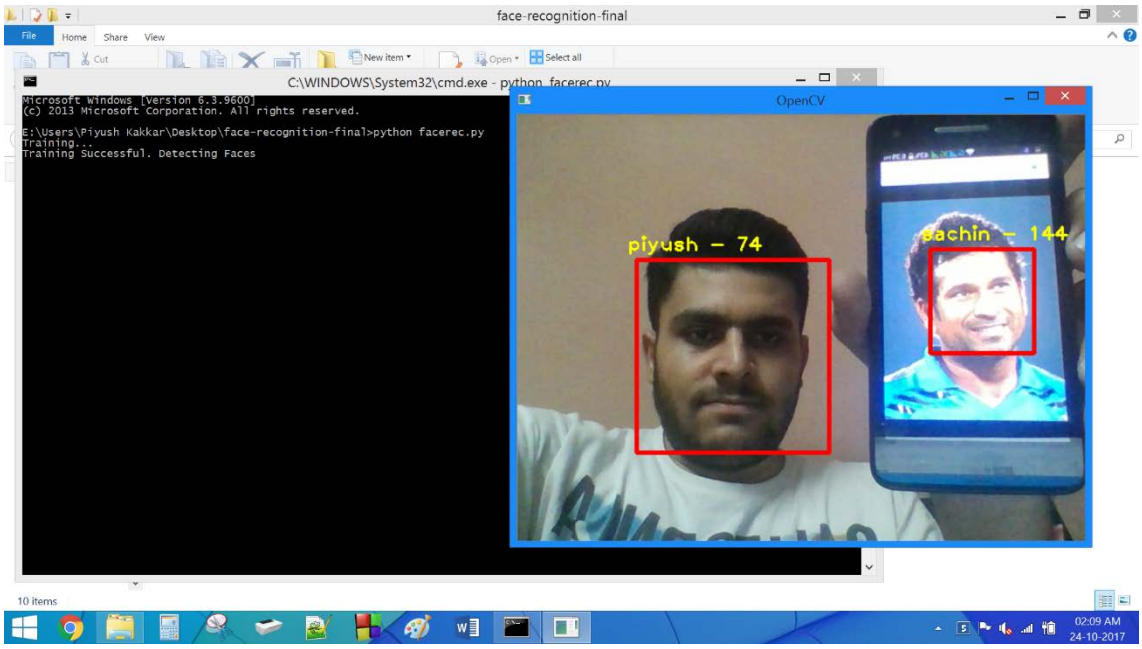

Figure 15: *Output: Multiple Face Detection*

Testing

8.1 Black Box Testing

1. Our model is capable of detecting and recognizing face of the person for which our model is trained along with the confidence level with which the face is recognized

   Lower the confidence value better is the face recognized.

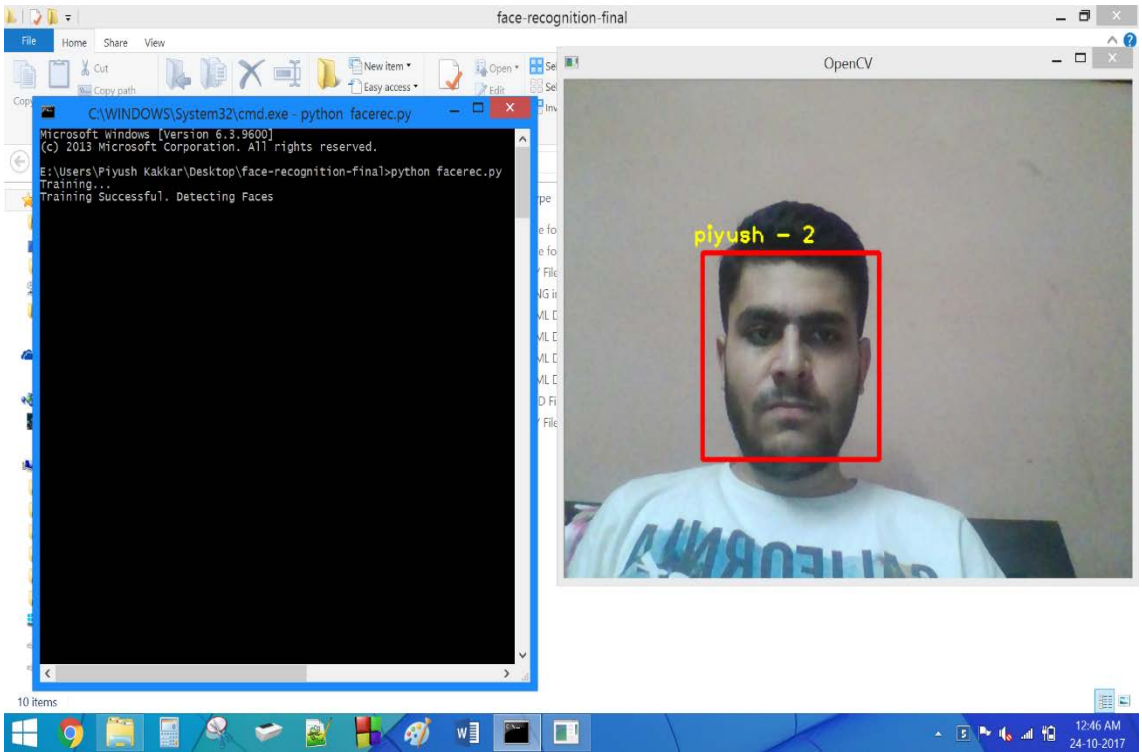

Figure 16: *Testing: Recognised Face*

2. Our model is also able to detect and recognize face of unknown person. When the confidence value for any detected face in the stream of images is greater than 500 then the face detected will be labelled as unknown.

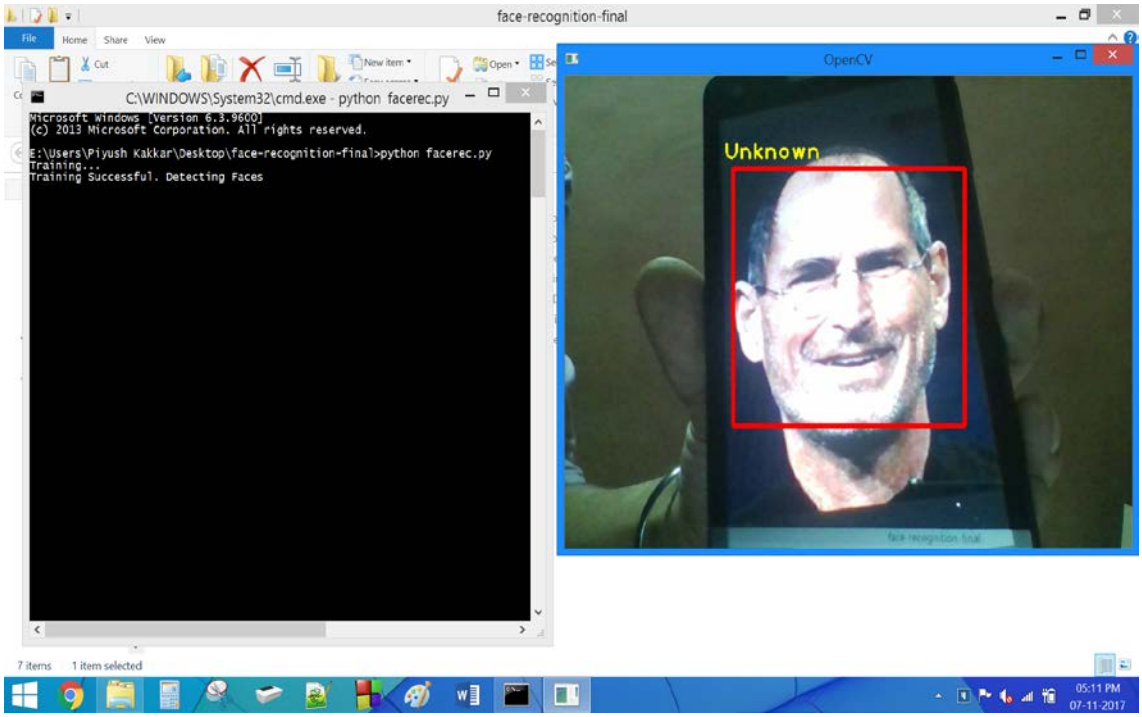

Figure 17: Testing: Unknown Face

3. The model is not able to recognize the face of the known person if the face in video stream is at certain angle or have some expressions.

   This problem could be rectified by training our model with faces of a person at different angles and with expression.

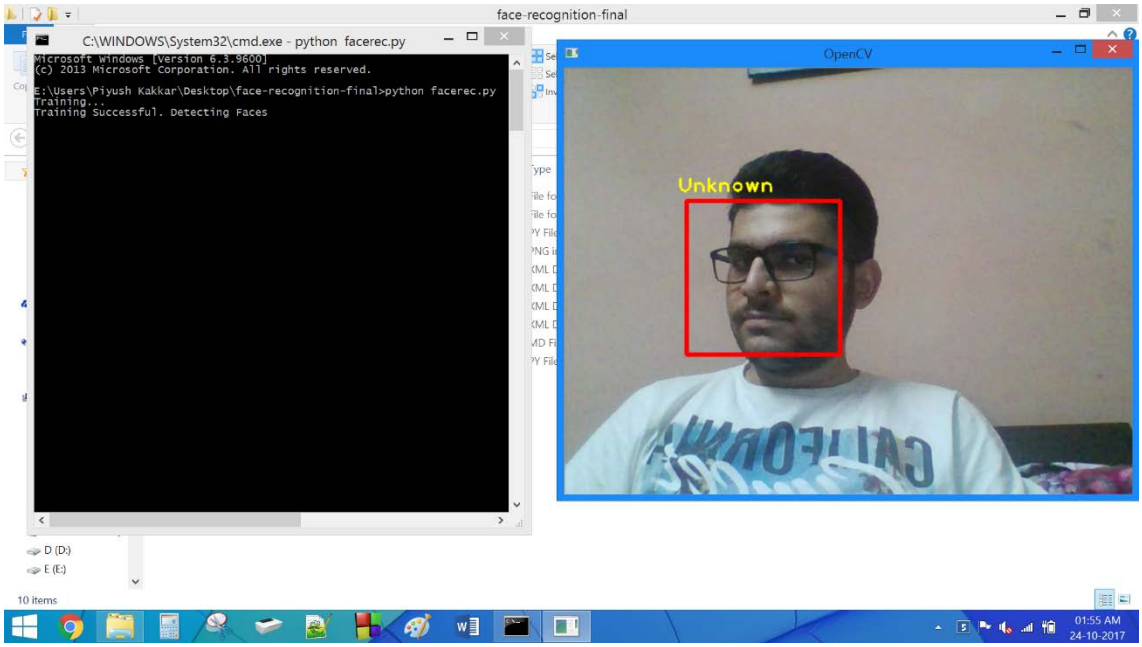

Figure 18: *Testing: Known face at an angle*

CONCLUSION

In this project, we are able to detect and recognize faces in real time obtained from camera. We have used Haar feature-based cascade classifiers approach for face detection. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Several advantages of this algorithm are: Efficient feature selection, Scale and location invariant detector, instead of scaling the image itself, we scale the features Such a generic detection scheme can be trained for detection of other types of objects (e.g. cars, legs). It also has some disadvantages: Detector is most effective only on frontal images of faces, it can hardly cope with 45° face rotation both around the vertical and horizontal axis and Sensitive to lighting conditions

FUTURE ENHANCEMENTS

With the successful detection of facial features, the next goal is to research the ability for more precise details, like individual points, of the facial features to be gathered. These points will be use to differentiate general human emotions, like happiness and sadness. Recognition of human emotion would require detection and analysis of the various elements of a human face, like the brow and the mouth, to determine an individual's current expression. The expression can then be compared to what is considered to be the basic signs of an emotion. This research will be used in the field human-computer interaction to analyse the emotions one exhibits while interacting with a user interface.

Furthermore, we will add the functionality of recognizing the gender of the people. We will also connect our face recognition system with a database in order to create a criminal identification system.

REFERENCES

1. Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. IEEE Transactions on Pattern Analysis and Machine Intelligence. 19, pp. 711-720. IEEE Computer Society
.
2. Bornet, O. (2005, May 19). Learning Based Computer Vision with Intel's Open Source Computer Vision Library. Retrieved April 2007, 2007, from Intel.com Web site: http://www.intel.com/technology/itj/2005/volume09issue02/art03_learning_vision/p04_face_dete ction.htm

3. Brunelli, R., & Poggio, T. (1993). Face Recognition: Features versus templates. IEEE Transaction on Pattern Analysis and Machine Intelligence , 15 (10), 1042-1052.

4. Choi, J., Lee, S., Lee, C., & Yi, J. (2001). A Real-time Face Recognition System using Multiple Mean Faces and Dual Mode Fisherfaces. IEEE Transactions on Pattem Analysis and machine Intelligence (pp. 1686-1689). Suwon, Korea: IEEE Computer Society.

5. Cristinacce, D. and Cootes, T. Facial feature detection using AdaBoost with shape constraints. British Machine Vision Conference, 2003.

6. Open Computer Vision Library Reference Manual. Intel Corporation, USA, 2001.

7. Viola, P. and Jones, M. Rapid object detection using boosted cascade of simple features. IEEE Conference on Computer Vision and Pattern Recognition, 2001.

8. http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

9. http://opencv-python-tutroals.readthedocs.io/en/latest/index.html

10.http://ieeexplore.ieee.org/document/7800424/

11.https://www.cs.auckland.ac.nz/~m.rezaei/Downloads.html