Attendance Posting Using

Face Recognition

Personal Project Report

by

K. Syam varaprasad Naidu

Under the Guidance

of

Dr.T.HitendraSarma

Srinivasa Ramanujan Inistitute of Technology

(Approved by AICTE New Delhi | Affiliated to JNTUA, Anantapuram |
Accredited by NAAC "A" Grade)

SRINIVASA RAMANUJAN INSTITUTE OF TECHNOLOGY

(Approved by AICTE, New Delhi | Affiliated to JNTUA, Ananthapuramu | Accredited by NAAC with 'A' grade)
Rotarypuram Village, B K Samudram Mandal, Ananthapuramu District, A.P. – 515 701.

CERTIFICATE

This is to certify that this project entitled "**Attendance posting using Face-recognition**" submitted to Department of Computer Science and Engineering, is a bonafide record of work done by **K.Syam varaprasad naidu** under the guidance of **Dr.T.HitendraSarma** during the academic year 2017-2018.

DR.T.HitendraSarma M.Tech.,Pd.D
(Head of Research & Development cell)

Dr.G.K.Venkata Narasimha Reddy Ph.D,
(Head of Computer Science Department)

Abstract

In today's networked world, one of the most successful application of image analysis and understanding, face recognition has recently received significant attention, especially during the past few years. Facial recognition technology has emerged as an attractive solution to address many contemporary needs of identification and verification of identity claims. It brings together the promise of other biometric systems, which attempt to identify the individual distinctive features of the body, and the more familiar functionality of visual surveillance systems. This project develops a facial recognition machine that can deploy an easy way of taking down attendance. This machine first captures the images of all authorized persons and stores into database. The machine then stores the image by mapping into a face coordinate structure. Next time whenever the person enters the premises then the machine marks the attendance along with date and time. These details are stored into database. In this machine contains Local Binary Patterns Histograms (LBPH) face recognizer.

Table of Contents

Introduction

Manual attendance taking can be quick easy process for small group of people when compared to large group of people it can become a cumbersome practice. To address this problem, face recognition is one of the solution. Face recognition is one of the few biometric methods that process the merits of both high accuracy of a psychological approach and low intrusiveness. For this reason, since the early 70's (Kelly), face recognition has drawn the attention of researchers in fields from security, psychology, and image processing to computer vision. when we come to the Attendance posting using face recognition others have addressed the problem using different recognition approaches like using Eigen faces, Fisher faces and Local Binary Patterns Histograms (LBPH). The subsequent chapters explains about these methods briefly and it's implementation. These are implemented using python, opencv and numpy.

Approach Used

For the problem attendance posting using face recognition, We know that *Eigenfaces* and *Fisherfaces* are both affected by light and in real life, we can't guarantee perfect light conditions. Local Binary Patterns Histograms (LBPH) is an improvement to overcome this draw back. The idea with LBPH is not look at the whole, but instead, try to find it's local structure by comparing each pixel to neighbouring pixels.

THE LBPH FACE RECOGNIZER PROCESS : Take a 3×3 window and move it across one image. At each move (each local part of the picture), compare the pixel at the center, with its surrounding pixels. Denote the neighbors with intensity value *less than or equal to* the center pixel by *1* and the rest by 0.

After you read these 0/1 values under the 3×3 window in a clockwise order, you will have a binary pattern like *11100011* that is local to a particular area of the picture. When you finish doing this on the whole image, you will have a list of **local binary patterns**.
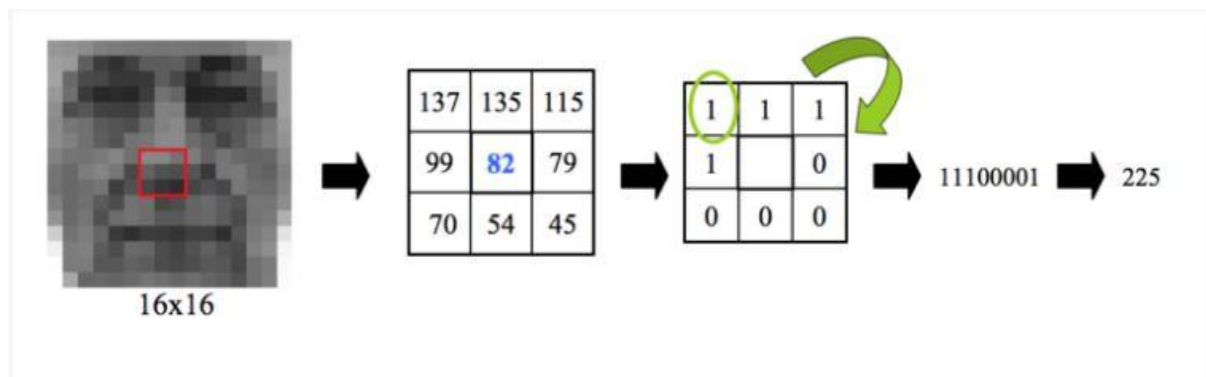

Figure: 2.1

Now, after you get a list of local binary patterns, you convert each one into a decimal number using **binary to decimal conversion** (as shown in above image) and then you make a **histogram** of all of those decimal values. A sample histogram looks like this:
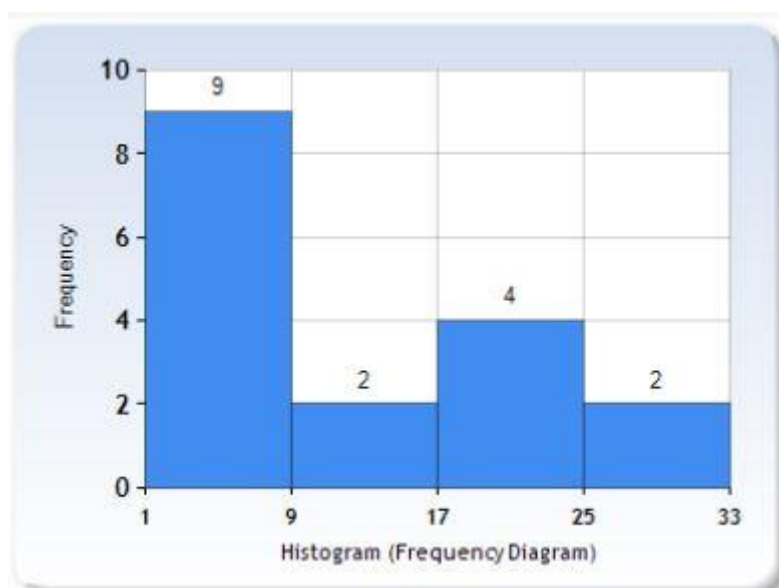

Figure: 2.2

In the end, you will have one histogram for each face in the training data set. That means that if there were 100 images in the training data set then LBPH will extract 100 histograms after training and store them for later recognition. Remember, the **algorithm also keeps track of which histogram belongs to which person**.

Later during recognition, the process is as follows:

1. Feed a new image to the recognizer for face recognition.

2. The recognizer generates a histogram for that new picture.

3. It then compares that histogram with the histograms it already has.

4. Finally, it finds the best match and returns the person label associated with that best match.

Below is a group of faces and their respective local binary patterns images. You can see that the **LBP faces are not affected by changes in light conditions**:
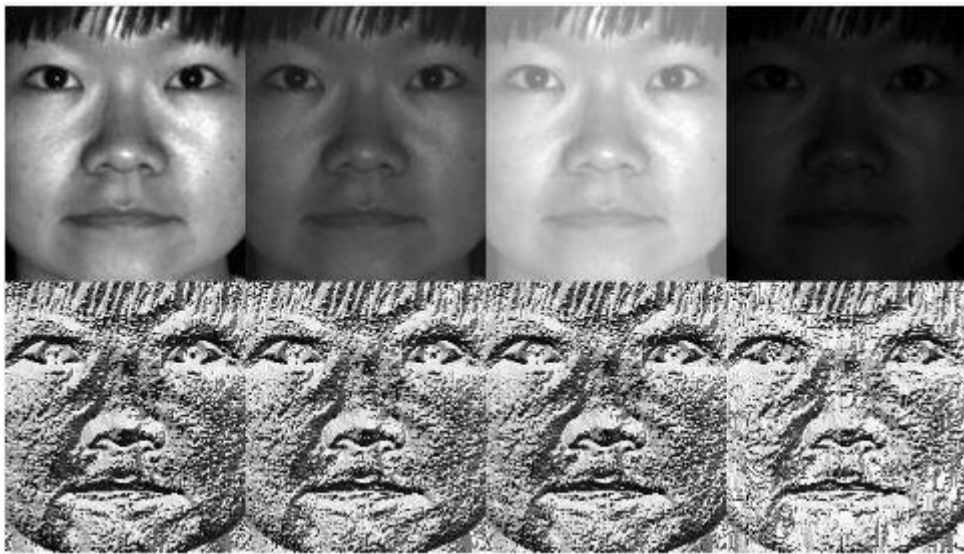

Figure: 2.3

Theory of Opencv Face Recognizers

Thanks to OpenCV, coding facial recognition is now easier than ever. There are three easy steps to computer coding facial recognition, which are similar to the steps that our brains use for recognizing faces. These steps are:

1. **Prepare Training Data:** Read training images for each person/subject along with their labels, detect faces from each image and assign each detected face an **integer label** of the person it belongs.

2. **Train Face Recognizer:** Train OpenCV's LBPH recognizer by feeding it the data we prepared in step 1.

3. **Prediction:** Introduce some test images to face recognizer and see if it predicts them correctly.

As we discussed before, OpenCV has three built-in face recognizers and thanks to its clean coding, you can use any of them just by changing a single line of code. Here are the names of those face recognizers and their OpenCV calls:

- **EigenFaces** – cv2.face.createEigenFaceRecognizer()
- **FisherFaces** – cv2.face.createFisherFaceRecognizer()
- **Local Binary Patterns Histograms (LBPH)** –cv2.face.createLBPHFaceRecognizer()

Even though LBPH Face Recognizer is efficient than eigen and fisher face recognizers, let us see them in brief:

EIGENFACES FACE RECOGNIZER :

This algorithm considers the fact that **not all parts of a face are equally important or useful for face recognition**. Indeed, when you look at someone, you recognize that person by his distinct features, like the eyes, nose, cheeks or forehead; and how they vary respect to each other.

In that sense, you are focusing on the *areas of maximum change*. For example, from the eyes to the nose there is a significant change, and same applies from the nose to the mouth. When you look at multiple faces, you compare them by looking at these areas, because by catching the maximum variation among faces, they help you differentiate one face from the other.

In this way, is how EigenFaces recognizer works. It looks at all the training images of all the people *as a whole* and tries to extract the components which are relevant and useful and discards the rest. These important features are called **principal components**.

Note: We will use the terms: *principal component*s, *variance*, *areas of high change* and *useful features* indistinctly as they all mean the same.

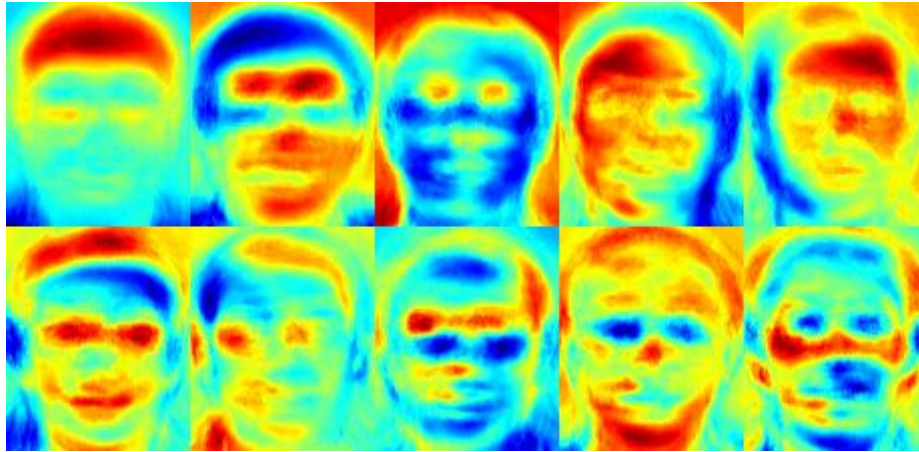Below is an image showing the variance extracted from a list of faces.

Figure :

EigenFaces Face Recognizer Principal Components. Source: **docs.opencv.org**
You can see that the useful features represent faces which receive the name of *Eigen Faces*. I mean, *how else do you think the algorithm got its name?*

So, EigenFaces recognizer trains itself by extracting principal components, but it also **keeps a record** of which ones belong to which person. Thus, whenever you introduce a new image to the algorithm, it repeats the same process as follows:

1. Extract the principal components from the new picture.
2. Compare those features with the list of elements stored during training.
3. Find the ones with the best match.
4. Return the 'person' label associated with that best match component.

In simple words, it's a game of matching.

However, one thing to note in above image is that **EigenFaces algorithm also considers illumination as an important feature**. In consequence, lights and shadows are picked up by EigenFaces, which classifies them as representing a 'face.'

Face recognition picks up on human things, dominated by shapes and shadows: two eyes, a nose, a mouth.

FISHERFACES FACE RECOGNIZER :

This algorithm is an improved version of the last one. As we just saw, **EigenFaces** looks at all the training faces of all the people at once and finds principal components from all of them combined. By doing that, it doesn't focus on the features that discriminate one individual from another. Instead, it concentrates on the ones that represent all the faces of all the people in the training data, *as a whole*.
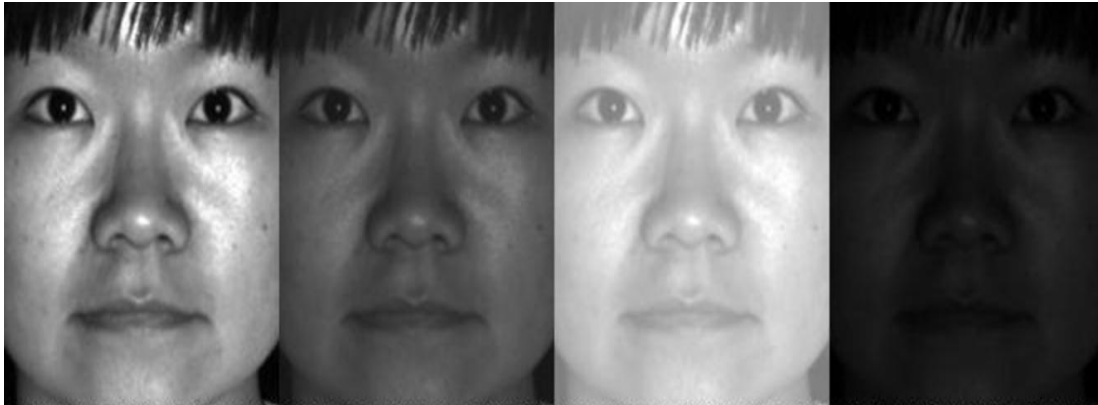
But here's the kicker:

Consider the lighting changes in following images:

Figure:

Since EigenFaces also finds illumination as a useful component, it will find this variation very relevant for face recognition and may discard the features of the other people's faces, considering them less useful. In the end, the variance that EigenFaces has extracted represents *just one individual's facial features*.

HOW TO FIX THIS ISSUE?

We can do it by tunning EigenFaces so that it extracts useful features from the faces of each person separately instead of extracting them from all the faces combined. In this way, even if one person has high illumination changes, it will not affect the other people's features extraction process.

Precisely, **FisherFaces** face recognizer algorithm extracts principal components that differentiate one person from the others. In that sense, an individual's components do not dominate (become more useful) over the others.

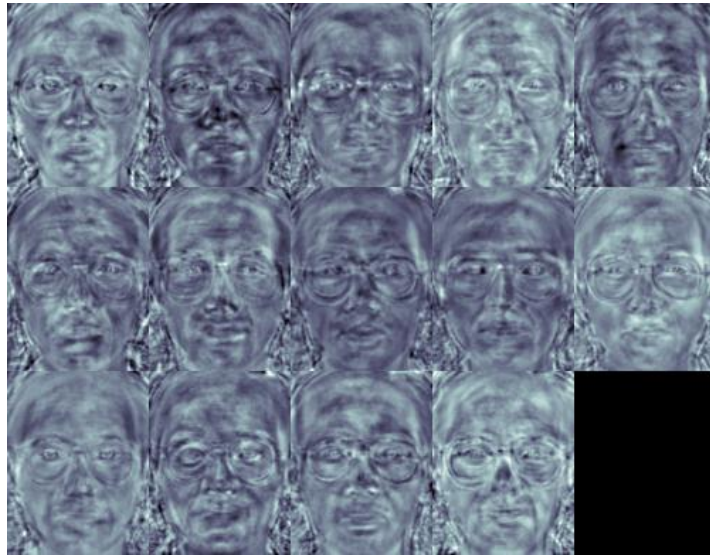Below is an image of principal components using FisherFaces algorithm.

Figure:

FisherFaces Face Recognizer Principal Components. Source: **docs.opencv.org**
You can see that the features represent faces which receive the name of *Fisher Faces. Are you noticing a theme with the names of the algorithms?*

One thing to note here is that FisherFaces only prevents features of one person from becoming dominant, but it still considers illumination changes as a useful feature. We know that light variation is not a useful feature to extract as it is not part of the actual face.

Installation

1. Opencv : Open Source Computer Vision Library: http://opencv.org is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

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

If you are familiar with python then you can use the command called 'pip' for the installation purpose of any module. Go to the command prompt then enter the command as shown below:
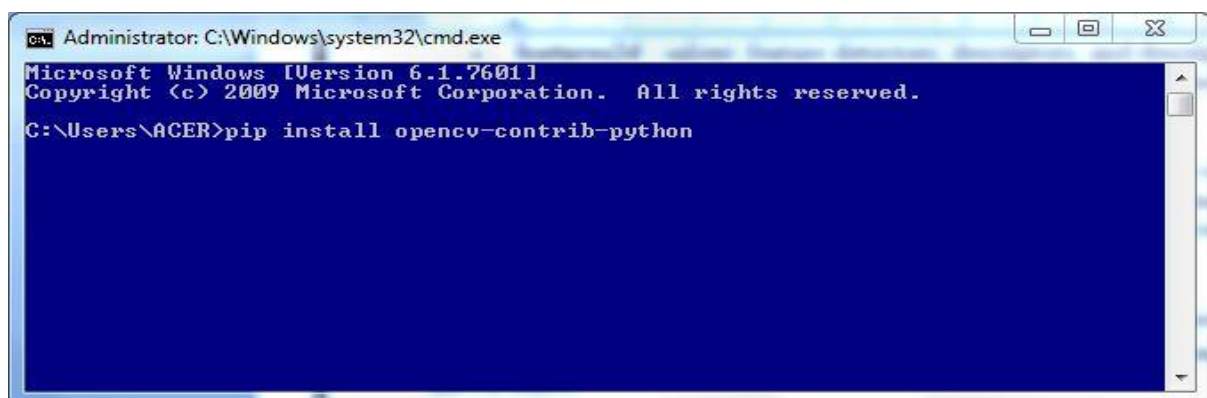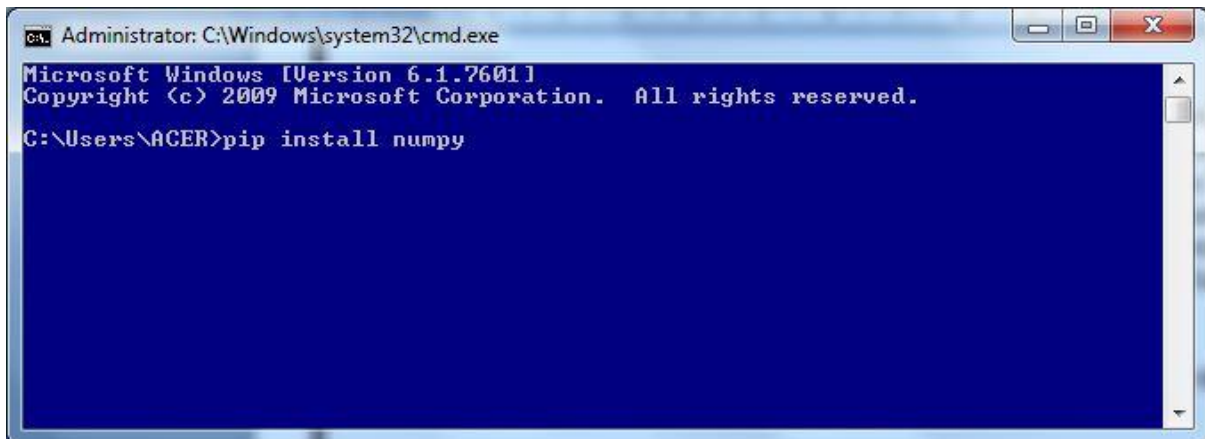

Figure: 3.1

Similarly you can use the same command for uninstalling the package by replacing 'install' with 'uninstall'

2. NumPy : NumPy is an acronym for "Numeric Python" or "Numerical Python". It is an open source extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches the programming language Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices. The implementation is even aiming at huge matrices and arrays. Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.

The NumPy module can be automatically installed while installing the opencv.if you want to install it the command is shown below:
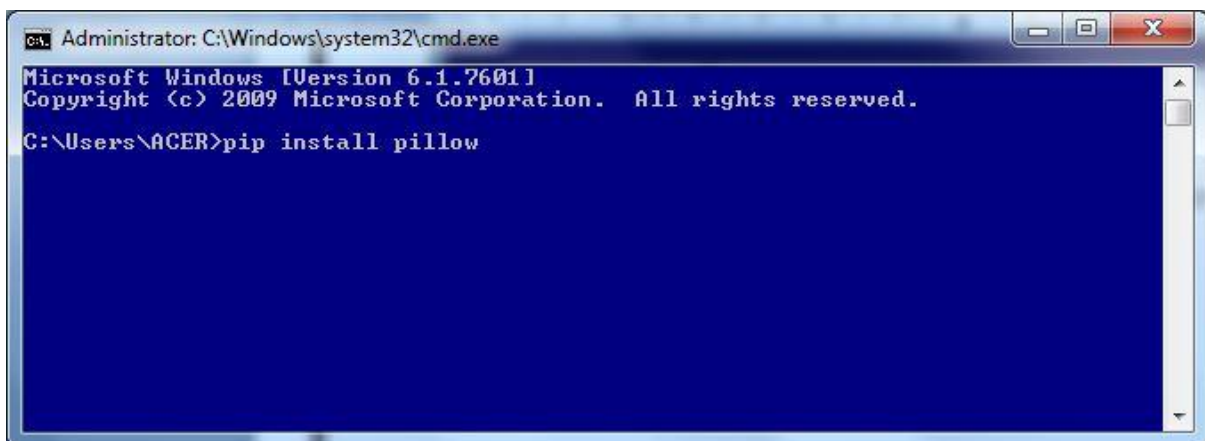
Figure**:** 3.2

3. PIL : Python Imaging Library(PIL) is a package that exposes many functions to manipulate images from a Python script. The command show below inthe figure is used to install python image library:

Figure: 3.3

4. SQLite studio: SQLite studio is a very easy to use database engine. It is self-contained, serverless, zero-configuration and transactional. It is very fast and lightweight, and the entire database is stored in a single disk file. It is used in a lot of applications as internal data storage. The Python Standard Library includes a module called "sqlite3" intended for working with this database. This module is a SQL interface compliant with the DB-API 2.0 specification. you can download with this link sqlitestudio. After downloading, extracting the file you can find the application.

Figure: 3.4

Basic Program on Face Detection

The below program explains a basic program on face detection :
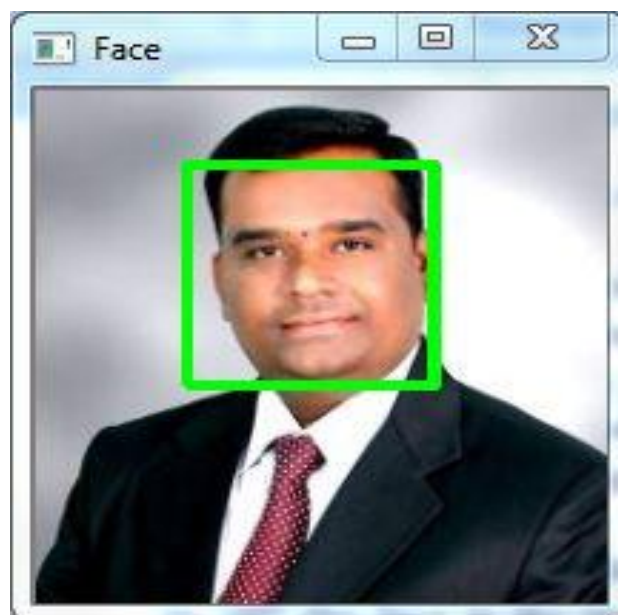
```
1    import cv2
2    import numpy as np
3
4
5    faceDetect=cv2.CascadeClassifier('haarcascade_frontalface_default.xml');
6
7    img = cv2.imread('sarma.jpg')
8    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
9    faces=faceDetect.detectMultiScale(gray,1.3,5)
10   while True:
11       gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
12       faces=faceDetect.detectMultiScale(gray,1.3,5)
13       for(x,y,w,h) in faces:
14           cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
15       cv2.imshow('Face',img)
16       if cv2.waitKey(20)==ord('q') :
17           break
18   cv2.destroyAllWindows()
```

Figure : 4.1

OpenCV comes with a trainer as well as detector. If you want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one. Its full details are given here: Cascade Classifier Training. Here we will deal with detection. OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in opencv/data/haarcascades/ folder. Let's create face detector with OpenCV. First we need to load the required XML classifiers. In 5th line we are loading a frontal face classifier. In 6th and 7th line we are reading an image and the image is converted into grayscale. From line 10th to 16th the face in the image is detected and show until you press 'q'. Finally 18th line destroy all windows present.

The output is :

Basic Program on Face and Eye Detection

The below program explains a basic program on face detection :

```python
1    import numpy as np
2    import cv2
3
4    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
5    eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
6
7    img = cv2.imread('sachin.jpg')
8    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9
10   while True:
11       gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
12       faces=faceDetect.detectMultiScale(gray,1.3,5)
13       for(x,y,w,h) in faces:
14           cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
15           roi_gray = gray[y:y+h, x:x+w]
16           roi_color = img[y:y+h, x:x+w]
17           eyes = eye_cascade.detectMultiScale(roi_gray)
18           for (ex,ey,ew,eh) in eyes:
19               cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
20       cv2.imshow('Face',img)
21       if cv2.waitKey(10)==ord('q') :
22           break
23   cv2.destroyAllWindows()
```

Figure : 5.1

As we discussed in previous program, this includes eye detection. In 4th and 5th line includes loading of frontal face and eye classifier. From 10th to 22nd line detects face and displays eyes until you press 'q'. At 23rd line destroy all windows.
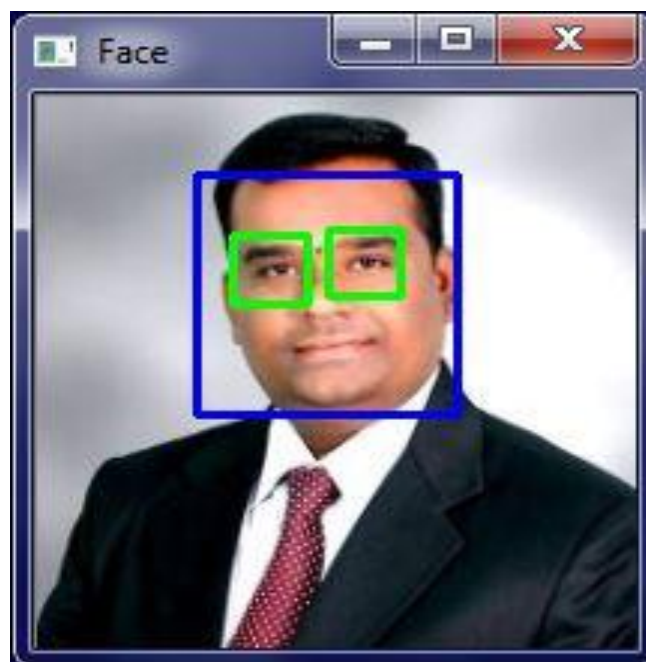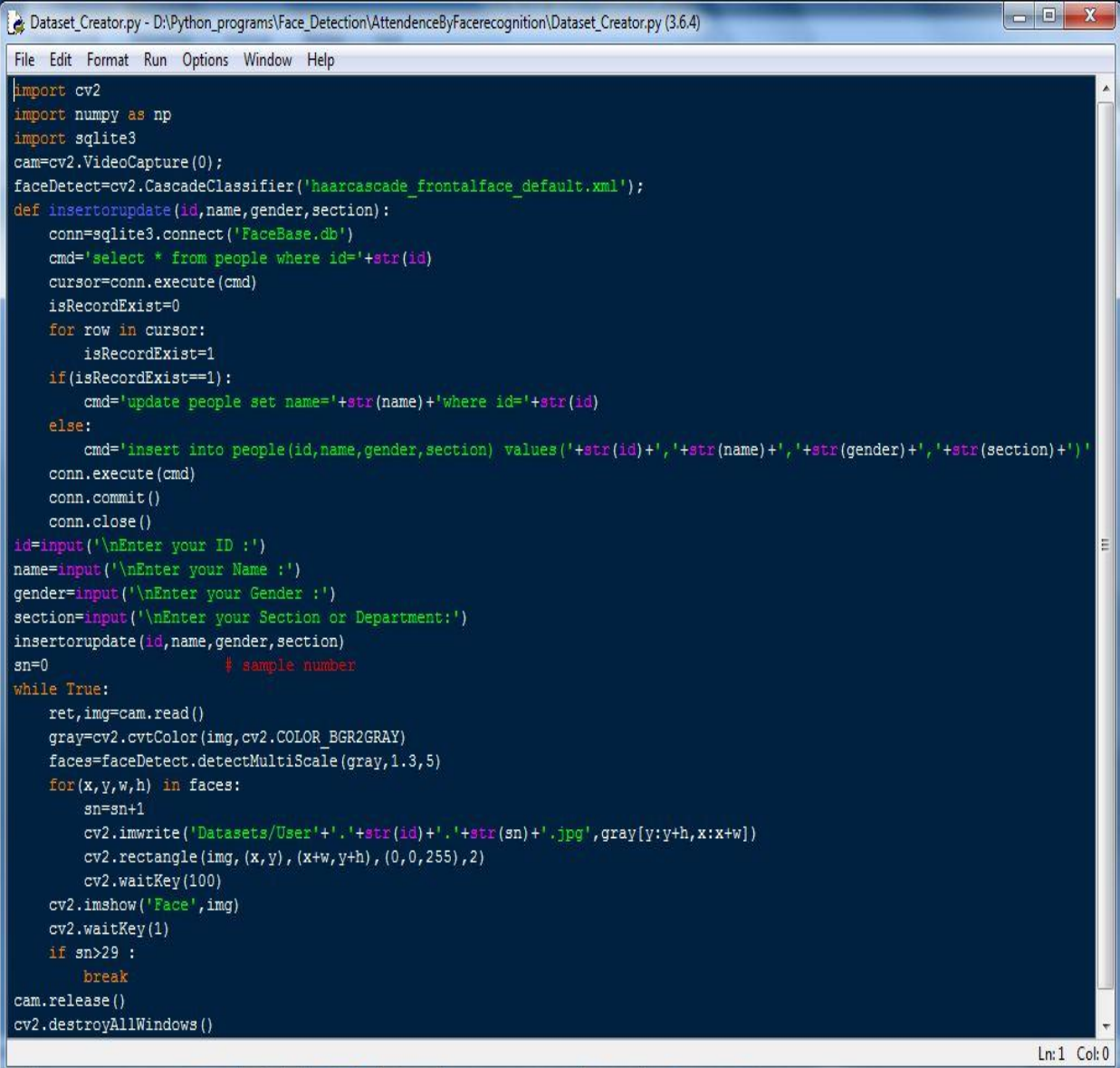
The output for the above code is :

Figure : 5.2

Creating a Dataset

The dataset creation means creating a file which includes faces of all authorized persons and details like ID, NAME, DEPARTMENT, ATTENDENCE STATUS are stores into database. This can be fulfilled using the SQLite .The below program explains about the dataset creation:

```
Dataset_Creator.py - D:\Python_programs\Face_Detection\AttendenceByFacerecognition\Dataset_Creator.py (3.6.4)

File  Edit  Format  Run  Options  Window  Help

import cv2
import numpy as np
import sqlite3
cam=cv2.VideoCapture(0);
faceDetect=cv2.CascadeClassifier('haarcascade_frontalface_default.xml');
def insertorupdate(id,name,gender,section):
    conn=sqlite3.connect('FaceBase.db')
    cmd='select * from people where id='+str(id)
    cursor=conn.execute(cmd)
    isRecordExist=0
    for row in cursor:
        isRecordExist=1
    if(isRecordExist==1):
        cmd='update people set name='+str(name)+'where id='+str(id)
    else:
        cmd='insert into people(id,name,gender,section) values('+str(id)+','+str(name)+','+str(gender)+','+str(section)+')'
    conn.execute(cmd)
    conn.commit()
    conn.close()
id=input('\nEnter your ID :')
name=input('\nEnter your Name :')
gender=input('\nEnter your Gender :')
section=input('\nEnter your Section or Department:')
insertorupdate(id,name,gender,section)
sn=0                       # sample number
while True:
    ret,img=cam.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces=faceDetect.detectMultiScale(gray,1.3,5)
    for(x,y,w,h) in faces:
        sn=sn+1
        cv2.imwrite('Datasets/User'+'.'+str(id)+'.'+str(sn)+'.jpg',gray[y:y+h,x:x+w])
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
        cv2.waitKey(100)
    cv2.imshow('Face',img)
    cv2.waitKey(1)
    if sn>29 :
        break
cam.release()
cv2.destroyAllWindows()
```

Figure : 6.1

The above program takes input consisting id, name, gender and department of a student or an authorized person. These details are stored into a database and the 30 facial images are captured and stored into a file . Each image has an label .In the above program the function insertorupdate() do the work ie.., writing the details of an authorized person into a database.

After giving input the resultant database and file with images are:

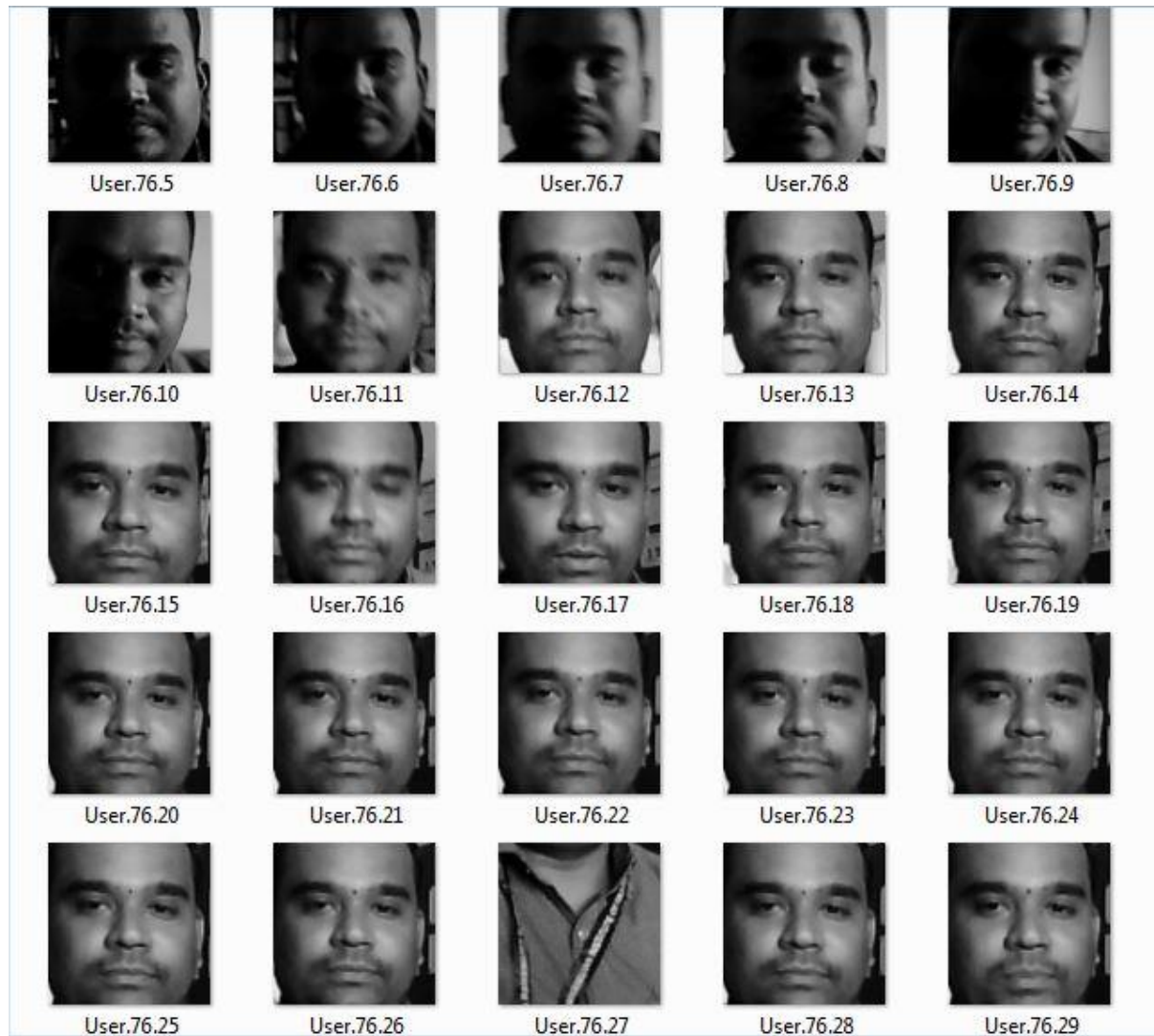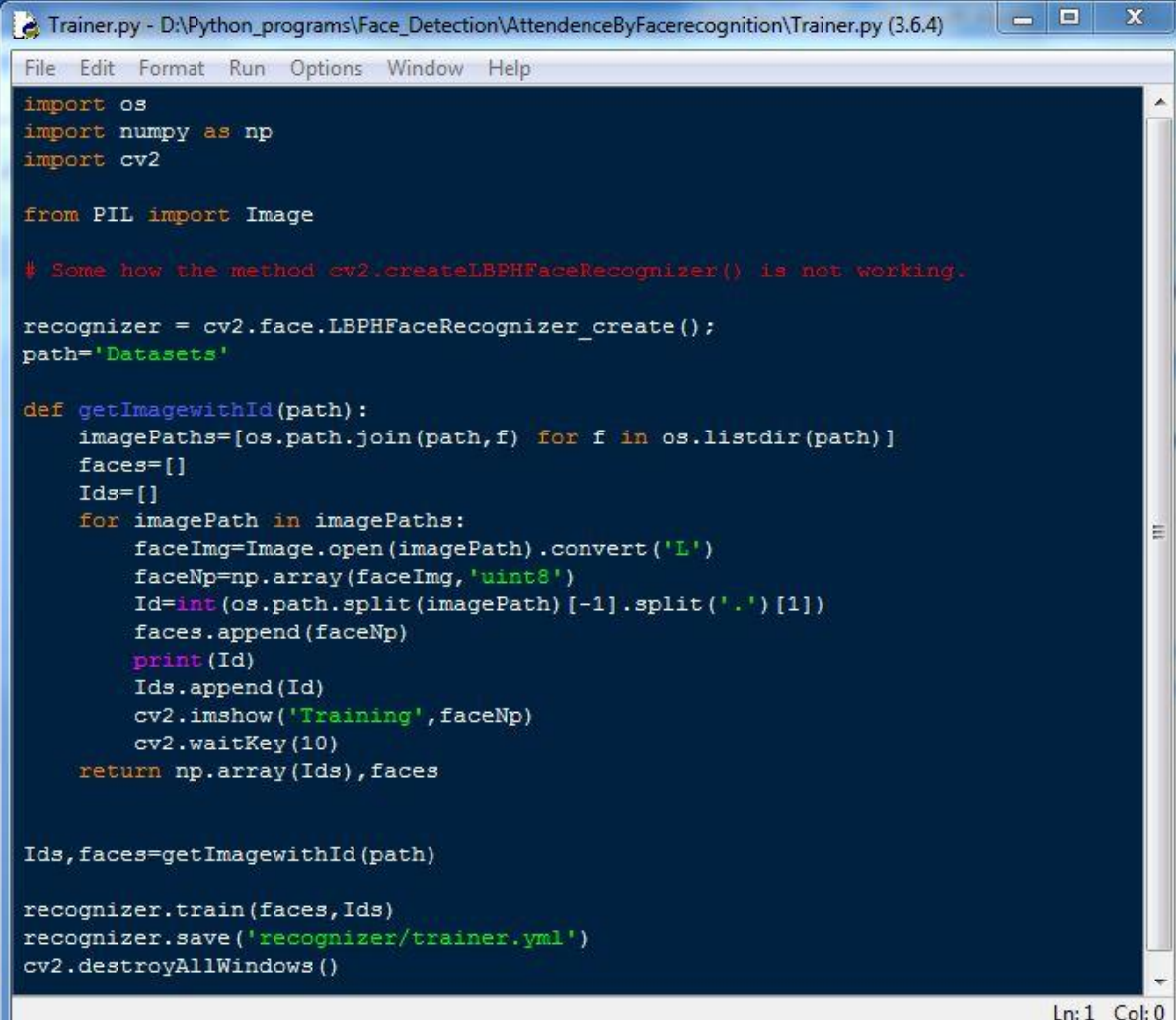|	id	name	gender	section	att
1	2	Aruna	Female	Unknown	NotEntered
2	1	SaiPriya	Female	Unknown	NotEntered
3	11	SyamKakarla	Male	CSE	NotEntered
4	589	Rahul	Male	CSE	NotEntered
5	76	HitendraSarma	Male	CSE	NotEntered
6	111	syam	male	CSE	NotEntered

Figure : 6.2

Figure: 6.3

Like this it captures the 30 images of an every individual.

Training the Dataset

After gathering of all data including facial images and the details of authorized persons, then we need to train the recognizer. For this the below program address the solution:

```python
import os
import numpy as np
import cv2

from PIL import Image

# Some how the method cv2.createLBPHFaceRecognizer() is not working.

recognizer = cv2.face.LBPHFaceRecognizer_create();
path='Datasets'

def getImagewithId(path):
    imagePaths=[os.path.join(path,f) for f in os.listdir(path)]
    faces=[]
    Ids=[]
    for imagePath in imagePaths:
        faceImg=Image.open(imagePath).convert('L')
        faceNp=np.array(faceImg,'uint8')
        Id=int(os.path.split(imagePath)[-1].split('.')[1])
        faces.append(faceNp)
        print(Id)
        Ids.append(Id)
        cv2.imshow('Training',faceNp)
        cv2.waitKey(10)
    return np.array(Ids),faces


Ids,faces=getImagewithId(path)

recognizer.train(faces,Ids)
recognizer.save('recognizer/trainer.yml')
cv2.destroyAllWindows()
```

Figure: 7.1

The method getImagewithId() takes path of the dataset consisting images as input and extracts the label of each image and stores into a numpy array and returns the array with labels and faces. By using the built in method of opencv ie.., recognizer.train( ) and the labels are stored into a file.

Detector

Initially we need to create two tables, one is for the attendance entry and the second one for the entry of date and time for the corresponding period. The resultant two tables after creation are :

1. The ATTENDENCE table is :

	id	name	p1	p2	p3	p4	p5	p6	p7
1	589	Rahul	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	11	Syamkakarla	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	76	HitendraSarma	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	2	Aruna	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	111	syam	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure: 8.1

The above table contains nine columns, two are id, name and seven are for respective periods initially these fields are set to null.
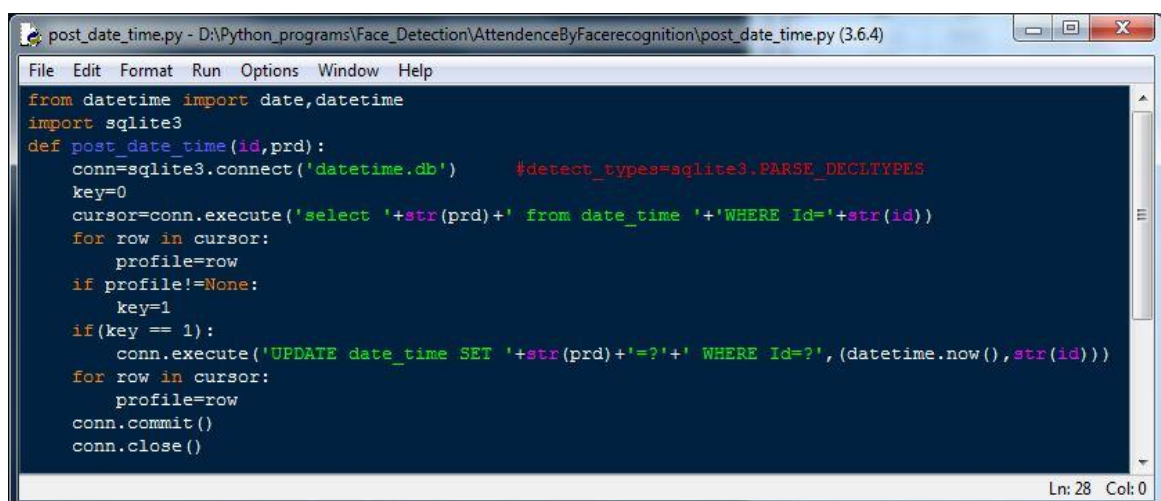
2. The DATE_TIME table is :

	id	p1	p2	p3	p4	p5	p6	p7
1	2	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	11	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	76	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	111	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	589	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure: 8.2

The above table contains eight columns, one is id and seven are for respective periods initially these fields are set to null. Next we need to discuss about the methods that i have written:

- post_date_time( ) :

```
from datetime import date,datetime
import sqlite3
def post_date_time(id,prd):
    conn=sqlite3.connect('datetime.db')        #detect_types=sqlite3.PARSE_DECLTYPES
    key=0
    cursor=conn.execute('select '+str(prd)+' from date_time '+'WHERE Id='+str(id))
    for row in cursor:
        profile=row
    if profile!=None:
        key=1
    if(key == 1):
        conn.execute('UPDATE date_time SET '+str(prd)+'=?'+' WHERE Id=?',(datetime.now(),str(id)))
    for row in cursor:
        profile=row
    conn.commit()
    conn.close()
```

Figure: 8.3

The above method takes id and period as input and insert the date and timestamp into the 'datetime' database.

- getprofileId( ) :

```
def getprofileId(id):
    conn=sqlite3.connect('FaceBase.db')
    cmd='select * from people where id='+str(id)
    cursor=conn.execute(cmd)
    profile=None
    for row in cursor:
        profile=row
    conn.close()
    return profile
```

Figure: 8.4

The above method takes id as input and if the person details are present in the database, it returns the row otherwise none is returned.

- postatt( ) :

```
def postatt(id,prd):
    conn=sqlite3.connect('attendenceBase.db')
    conn.execute('UPDATE attendence SET '+str(prd)+'=?'+'WHERE Id =?',('Present',id))
    conn.commit()
    conn.close()
```

Figure: 8.5

The above method takes id and period as input, then it updates the database as the respective person is present.

Next coming to the main program that identifies the person and post the attendance with date and time, which are stored in a database.

- Main program :

```
import cv2,os
import numpy as np
from PIL import Image
import pickle
import sqlite3
from datetime import date,datetime
from post_date_time import post_date_time
```

Figure: 8.6

Importing the necessary methods, where the datetime is used to take the current date and time of the system, sqlite3 is used to access the database and PIL is used to manipulate the images.

```python
faceDetect=cv2.CascadeClassifier('haarcascade_frontalface_default.xml');
cam=cv2.VideoCapture(0);
path='Datasets'

rec = cv2.face.LBPHFaceRecognizer_create();
rec.read('recognizer\\trainer.yml')
id=0
font=cv2.FONT_HERSHEY_SIMPLEX
prd=input("\nEnter the period(p) with number to post attendence .\nEx : p1\n")
while True:
    ret,img=cam.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces=faceDetect.detectMultiScale(gray,scaleFactor=1.2,minNeighbors=5,minSize=(100,100),flags=cv2.CASCADE_SCALE_IMAGE)
    for(x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
        id,conf=rec.predict(gray[y:y+h,x:x+w])
        profile=getprofileId(id)
        postatt(id,prd)
        post_date_time(id,prd)
        if profile != None:
            cv2.putText(img,'Hiii '+str(profile[1]),(x,y+h+30),cv2.FONT_HERSHEY_TRIPLEX,1,(255,144,30),2)
            cv2.putText(img,'Attendence Posted ',(x,y+h+60),cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(30,144,255),2)
            #cv2.putText(img,'At Date & Time :',d_t[0],(x,y+h+60),cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,69255),2)
    cv2.imshow('Face',img)
    if cv2.waitKey(20)==ord('q') :
        break
cam.release()
cv2.destroyAllWindows()
```

Figure: 8.7

By reading the trainer dataset and the period, the current face is detected by using a predefined method called faceDetect.detectMultiscale( ) . After recognising the face by using 'id' and period we can able to post the attendance and the date & time into the database.

Result

As the project is about attendance posting using face recognition, here i used database to store the student details into 3 different tables :

1. Student Details
2. Attendance Details
3. Timestamp Details

Firstly, It ask the period number from 1 to 7. Then the authorized person face is detected as shown in below figure:
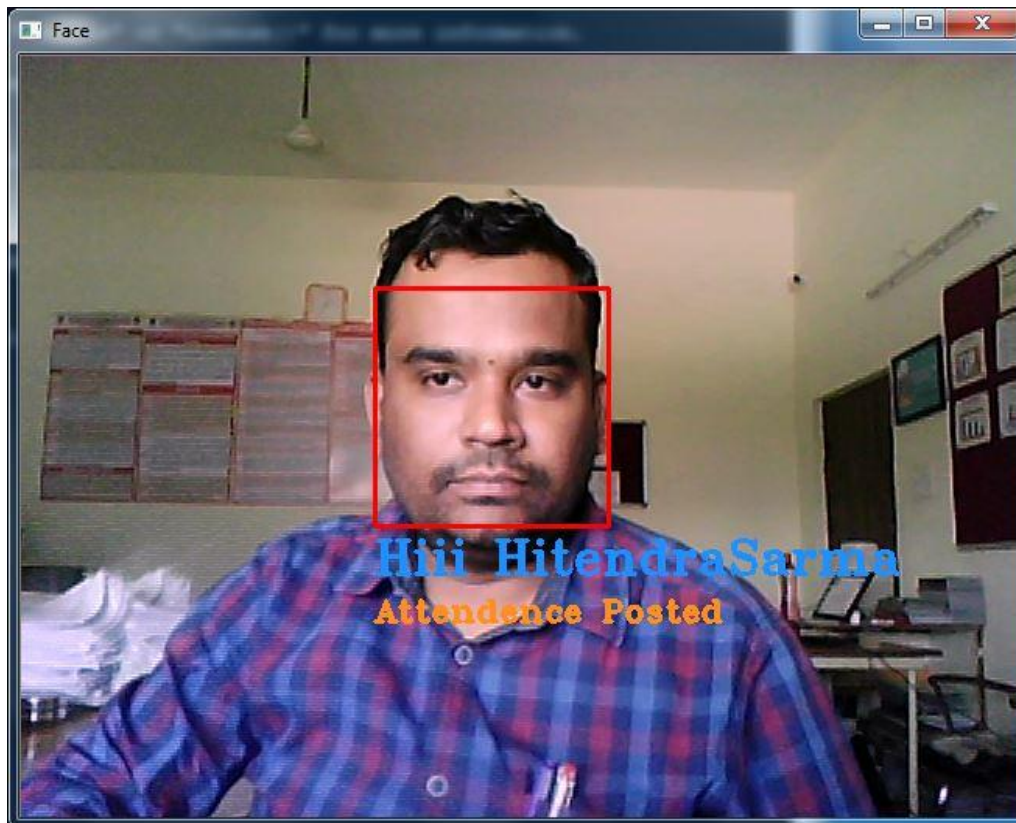

Figure:

It displays a message like "Hii --person name--" and "Attendence posted". Then the attendance and the respective timestamp is noted in the database. The below figures are of the respective database:

	id	name	p1	p2	p3	p4	p5	p6	p7
1	589	Rahul	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	11	Syamkakarla	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	76	HitendraSarma	NULL	Present	NULL	NULL	NULL	NULL	NULL
4	2	Aruna	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	111	syam	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure:

The timestamp with respect to the period is noted into the database :

id	p1	p2	p3	p4	p5	p6	p7
1	2	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	11	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	76	NULL	2018-03-13 13:34:34.899032	NULL	NULL	NULL	NULL	NULL
4	111	NULL	NULL	NULL	NULL	NULL	NULL	NULL
5	589	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure:

Once the attendance is taken to a particular period, the second time attendance cannot be accepted by the machine.

Conclusion

As this project is all about attendance posting using face recognition, The LBPH face - recognizer used in this project has some limitations and it does not have much accuracy when compared to the recognizers that implements using deep learning.

As per my results:

- It has 72.2% accuracy in order to identify the individual faces.
- It fails to recognize the correct person in the case of twins.
- It fails under poor light conditions.