FACE DETECTION AND RECOGNITION USING HAAR FEATURES AND LBPH ALGORITHM USING OPENCV

Submitted by

Tarun Sharma

Suriya Prakash

Abhishruti Mandal

In partial fulfillment of the requirements for the award of
Master Of Science
in Computer Science with Specialization in Data
Analytics Of

Cochin University of Science and Technology, Kochi

Conducted by

Indian Institute of Information Technology and Management-Kerala
Technopark Campus, Thiruvanathapuram-695 581

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to **Dr. Manoj Kumar T.K**, our project guide for providing us with the necessary facilities for the completion of this project. We are thankful for the valuable discussions we had at each phase of the project and for being a very supportive and encouraging project guide. We would also like to express our sincere and heartfelt gratitude to all people who were in one way or the other involved in my project. A lot of gratitude is reserved for the Director of the Institute for facilitating and nurturing an environment where I have been able to undertake this work. We would like to express my sincere thanks to all friends and classmates whose suggestions and creative criticism.

Name	Registration Number
Tarun Sharma	30009028
Suriya Prakash	30009199
Abhishruti Mandal	30010100

TABLE OF CONTENT

Abstract

Identifying a person with an image has been popularised 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 Data Analytics which is Elective for the second Semester. 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 Local binary pattern histograms were used. The methodology is described including flow charts for each stage of the system. Next, the results are shown followed by a discussion of encountered challenges. The report is concluded with the authors' opinion on the project and possible applications.

1.Introduction

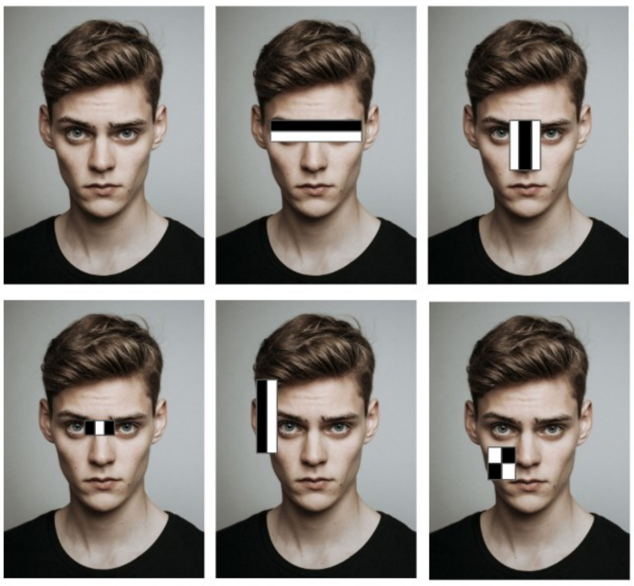
The following document is a report on the mini project for Robotic visual perception and autonomy. It involved building a system for face detection and face recognition using several classifiers available in the open computer vision library(OpenCV). Face recognition is a non-invasive identification system 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, Eigenface, Fisherface 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.

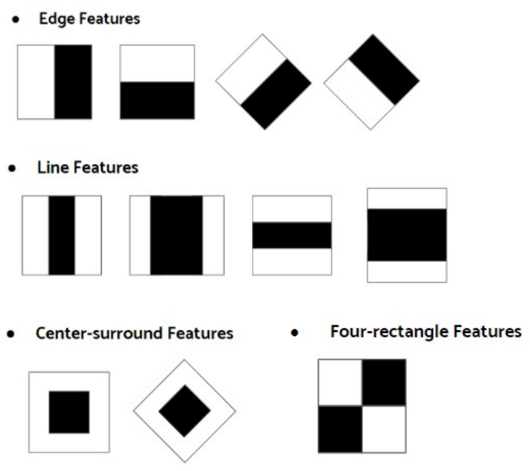
1.1 The History Of Face Recognition

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 [8], [9]. In This project, HAAR cascades are used for face detection and Eigenface, Fisherface and LBPH are used for face recognition.

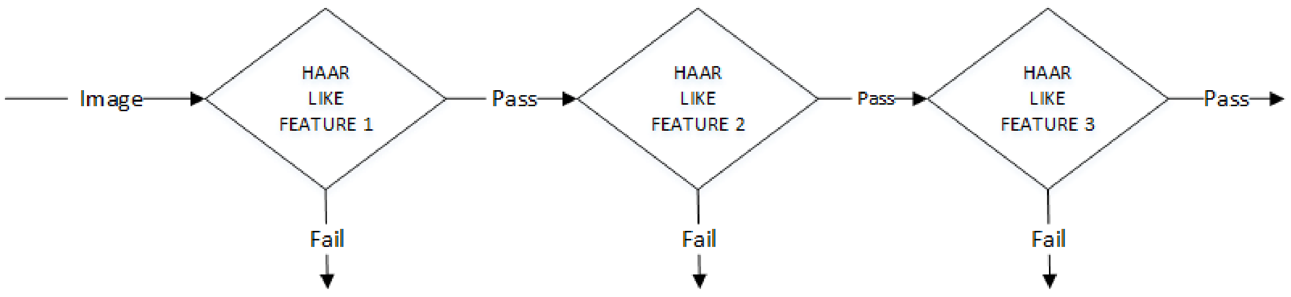
1.2 Face Detection using Haar-Cascade

DA Haar wavelet is a mathematical fiction that produces square-shaped waves with a beginning and an end and used to create box shaped patterns to recognise signals with sudden transformations. An example is shown in figure 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 analyse an image using Haar cascades, a scale is selected smaller than the target image. It is then placed on the image, and the average of the values of pixels in each section is taken. If the difference between two values pass a given threshold, it is considered a match. Face detection on a human face is performed by matching a combination of different Haar-like-features. For example, forehead, eyebrows and eyes contrast as well as the nose with eyes as shown below in figure A single classifier is not accurate enough. Several classifiers are combined as to provide an accurate face detection system as shown in the block diagram below in figure 3.





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 [7], 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 [7]. It can also reverse the direction of the classifier and get better results
if necessary [7]. 2. Furthermore, Weight-update-steps can be updated 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.



Haar Cascade Flow Chart

1.3 Face Recognition

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

Eigenface

Eigenface is based on PCA that classify images to extract features using a set of images. It is important that the images are in the same lighting condition and the eyes match in each image. Also, images used in this method must contain the same number of pixels and in grayscale. For this example, consider an image with n x n pixels as shown in figure 4. Each raw is concatenated to create a vector, resulting a 1×n 2 matrix. All the images in the dataset are stored in a single matrix resulting a matrix with columns corresponding the number of images. The matrix is averaged (normalised) to get an average human face. By subtracting the average face from each image vector unique features to each face are computed. In the resulting matrix, each column is a representation of the difference each face has to the average human face.The next step is computing the covariance matrix from the result. To obtain the Eigen vectors from the data, Eigen analysis is performed using principal component analysis. From the result, where covariance matrix is diagonal, where it has the highest variance is considered the 1st Eigen vector. 2nd Eigen vector is the direction of the next highest variance, and it is in 90 degrees to the 1st vector. 3rd will be the next highest variation, and so on. Each column is considered an image and visualised, resembles a face and called Eigenfaces. When a face is required to be recognised, the image is imported, resized to match the same dimensions of the test data as mentioned above. By projecting extracted features on to each of the Eigenfaces, weights can be calculated. These weights correspond to the similarity of the features extracted from the different image sets in the dataset to the features extracted from the input image. The input image can be identified as a face by comparing with the whole dataset. By comparing with each subset, the image can be identified as to which person it belongs to. By applying a threshold detection and identification can be controlled to eliminate false detection and recognition. PCA is sensitive to large numbers and assumes that the subspace is linear. If the same face is analysed under different lighting conditions, it will mix the values when distribution is calculated and cannot be effectively classified. This makes to different lighting conditions poses a problem in matching the features as they can change dramatically.

Fisherface

Fisherface technique builds upon the Eigenface and is based on LDA derived from Ronald Fishers' linear discriminant technique used for pattern recognition. However, it uses labels for classes as well as data point information [6]. When reducing dimensions, PCA looks at the greatest variance, while LDA, using labels, looks at an interesting dimension such that, when you project to that dimension you maximise the difference between the mean of the classes normalised by their variance [6]. LDA maximises the ratio of the between-class scatter and within-class scatter matrices. Due to this, different lighting conditions in images has a limited effect on the classification process using LDA technique. Eigenface maximises the variations while Fisherface maximises the mean distance between and different classes and minimises variation within classes. This enables LDA to differentiate between feature classes better than PCA and can be observed in figure 5 [12]. Furthermore, it takes less amount of space and is the fastest algorithm in this project. Because of these PCA is more suitable for representation of a set of data while LDA is suitable for classification.

Local Binary Pattern Histogram

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.It was first described in 1994 (LBP) and has since been found to be a powerful feature for texture classification. It has further been determined that when LBP is combined with histograms of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.Using the LBP combined with histograms we can represent the face images with a simple data vector.As LBP is a visual descriptor it can also be used for face recognition tasks, as can be seen in the following step-by-step explanation.

Step by Step:

Now that we know a little more about face recognition and the LBPH, let's go further and see the steps of the algorithm:

•**Parameters**: the LBPH uses 4 parameters:

•**Radius**: the radius is used to build the circular local binary pattern and represents the radius around the central pixel. It is usually set to

•**Neighbors**: the number of sample points to build the circular local binary pattern. Keep in mind: the more sample points you include, the higher the computational cost. It is usually set to 8.

•**Grid X**: the number of cells in the horizontal direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

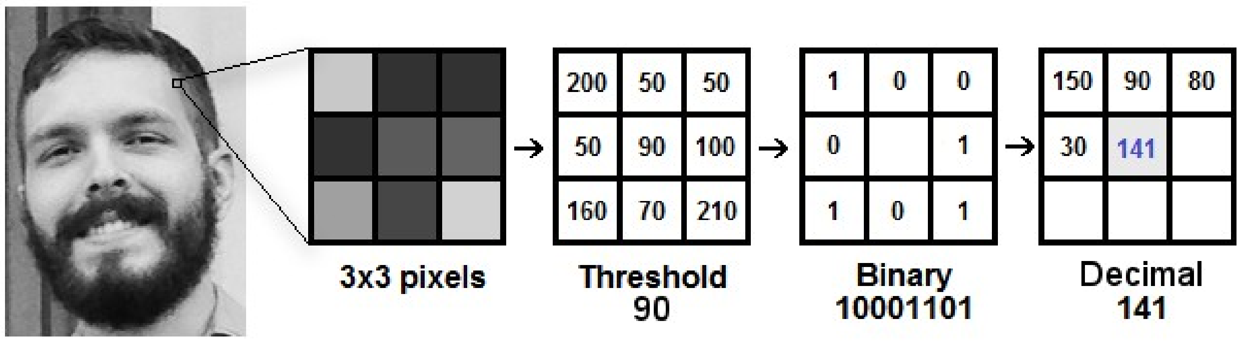
•**Grid Y**: the number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

Don't worry about the parameters right now, you will understand them after reading the next steps.

2. Training the Algorithm: First, we need to train the algorithm. To do so, we need to use a dataset with the facial images of the people we want to recognize. We need to also set an ID (it may be a number or the name of the person) for each image, so the algorithm will use this information to recognize an input image and give you an output. Images of the same person must have the same ID. With the training set already constructed, let's see the LBPH computational steps.

3. Applying the LBP operation: The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameters **radius** and **neighbors**.

The image below shows this procedure:



Based on the image above, let’s break it into several small steps so we can understand it easily:

•Suppose we have a facial image in grayscale.

•We can get part of this image as a window of 3x3 pixels.

•It can also be represented as a 3x3 matrix containing the intensity of each pixel (0~255).

•Then, we need to take the central value of the matrix to be used as the threshold.

•This value will be used to define the new values from the 8 neighbors.

•For each neighbor of the central value (threshold), we set a new binary value. We set 1 for values equal or higher than the threshold and 0 for values lower than the threshold.

•Now, the matrix will contain only binary values (ignoring the central value). We need to concatenate each binary value from each position from the matrix line by line into a new binary value (e.g. 10001101). Note: some authors use other approaches to concatenate the binary values (e.g. clockwise direction), but the final result will be the same.

•Then, we convert this binary value to a decimal value and set it to the central value of the matrix, which is actually a pixel from the original image.

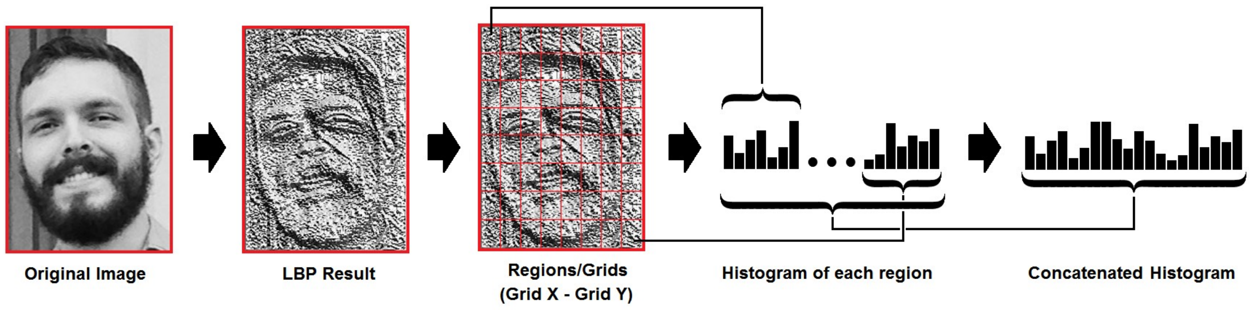
•At the end of this procedure (LBP procedure), we have a new image which represents better the characteristics of the original image.

•**Note**: The LBP procedure was expanded to use a different number of radius and neighbors, it is called Circular LBP.

It can be done by using **bilinear interpolation**. If some data point is between the pixels, it uses the values from the 4 nearest pixels (2x2) to estimate the value of the new data point.

4. Extracting the Histograms: Now, using the image generated in the last step, we can use the **Grid X** and **Grid Y** parameters to divide the image into multiple grids, as can be seen in the following image:

Based on the image above, we can extract the histogram of each region as follows:



•As we have an image in grayscale, each histogram (from each grid) will contain only 256 positions (0~255) representing the occurrences of each pixel intensity.

•Then, we need to concatenate each histogram to create a new and bigger histogram. Supposing we have 8x8 grids, we will have 8x8x256=16.384 positions in the final histogram. The final histogram represents the characteristics of the image original image. The LBPH algorithm is pretty much it.

5. Performing the face recognition: In this step, the algorithm is already trained. Each histogram created is used to represent each image from the training dataset. So, given an input image, we perform the steps again for this new image and creates a histogram which represents the image.

•So to find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram.

•We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: **euclidean distance**, **chi-square**, **absolute value**, etc. In this example, we can use the Euclidean distance (which is quite known) based on the following formula:

$$D = \sqrt{\sum_{i=1}^{n}(hist1_i - hist2_i)^2}$$

•So the algorithm output is the ID from the image with the closest histogram. The algorithm should also return the calculated distance, which can be used as a '**confidence**' measurement. **Note**: don't be fooled about the 'confidence' name, as lower confidences are better because it means the distance between the two histograms is closer.

•We can then use a threshold and the 'confidence' to automatically estimate if the algorithm has correctly recognized the image. We can assume that the algorithm has successfully recognized if the confidence is lower than the threshold defined.

Conclusions

•LBPH is one of the easiest face recognition algorithms.

•It can represent local features in the images.

•It is possible to get great results (mainly in a controlled environment).

•It is robust against monotonic gray scale transformations.

•It is provided by the OpenCV library (Open Source Computer Vision Library).

2. Methodology

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

2.1 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 eyeclassifier 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.

2.2 Face Recognition Process

For this project three algorithms are implemented independently. These are Eigenface, Fisherface and Linear binary pattern histograms respectively. All three can be implemented using OpenCV libraries. There are three stages for the face recognition as follows: 1. Collecting images IDs 2. Extracting unique features, classifying them and storing in XML files 3. Matching features of an input image to the features in the saved XML files and predict identity

2.3 Collection of image data

It is done using the Webcam only and store the images with the particular ids in the particular folder known as data here and the database of images is collected through the file called : **face_datasets.py**

3.Training the Classifier

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.createLBPHFaceRecognizer()

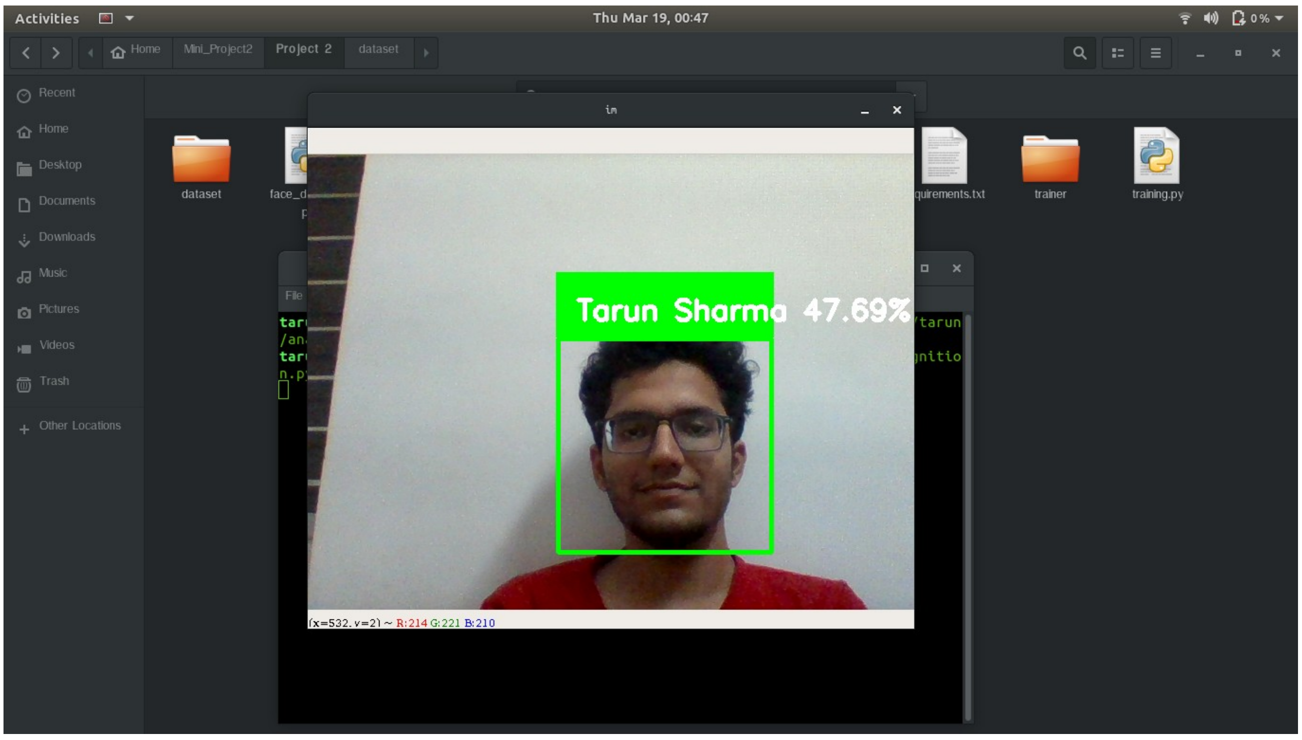
1. The radius from the centre pixel to build the local binary pattern.
2. The Number of sample points to build the pattern. Having a considerable number will slow down the computer.
3. The Number of Cells to be created in X axis.
4. The number of cells to be created in Y axis.
5. A threshold value similar to Eigenface and Fisherface. if the threshold is passed the object will return -1 Recogniser objects are created and images are imported, resized, converted into numpy arrays and stored in a vector. The ID of the image is gathered from splitting the file name, and stored in another vector. By using FaceRecognizer.train(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.

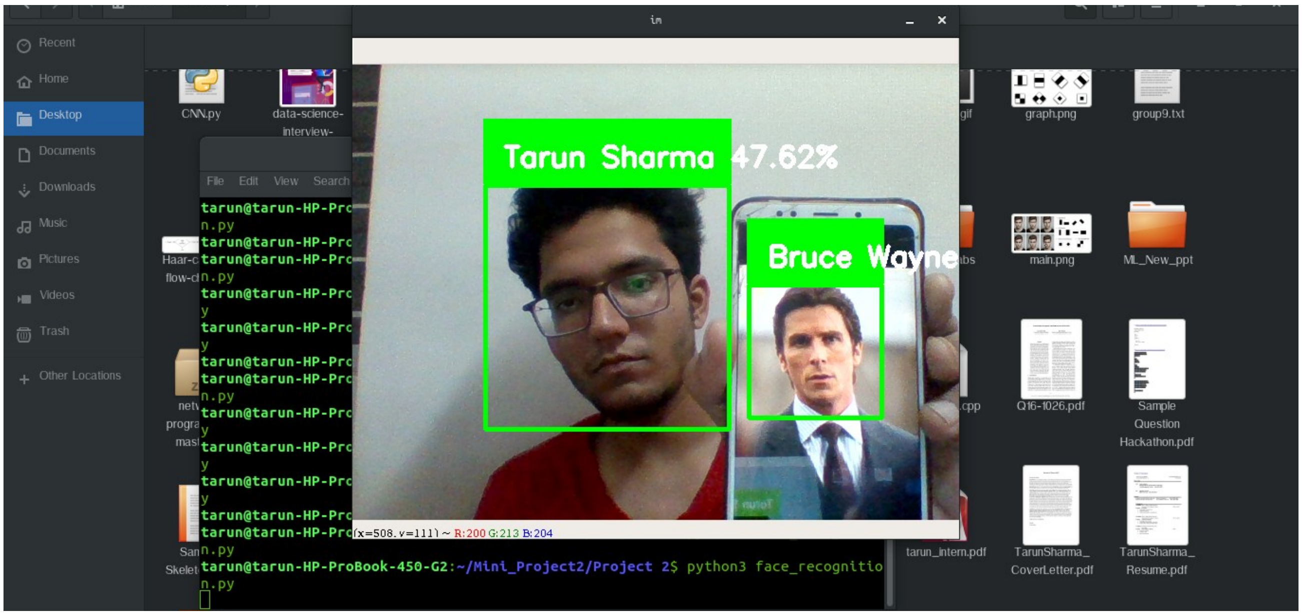
3.1 Face Recognition through classifier

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

4.<u>Results And Conclusion</u>

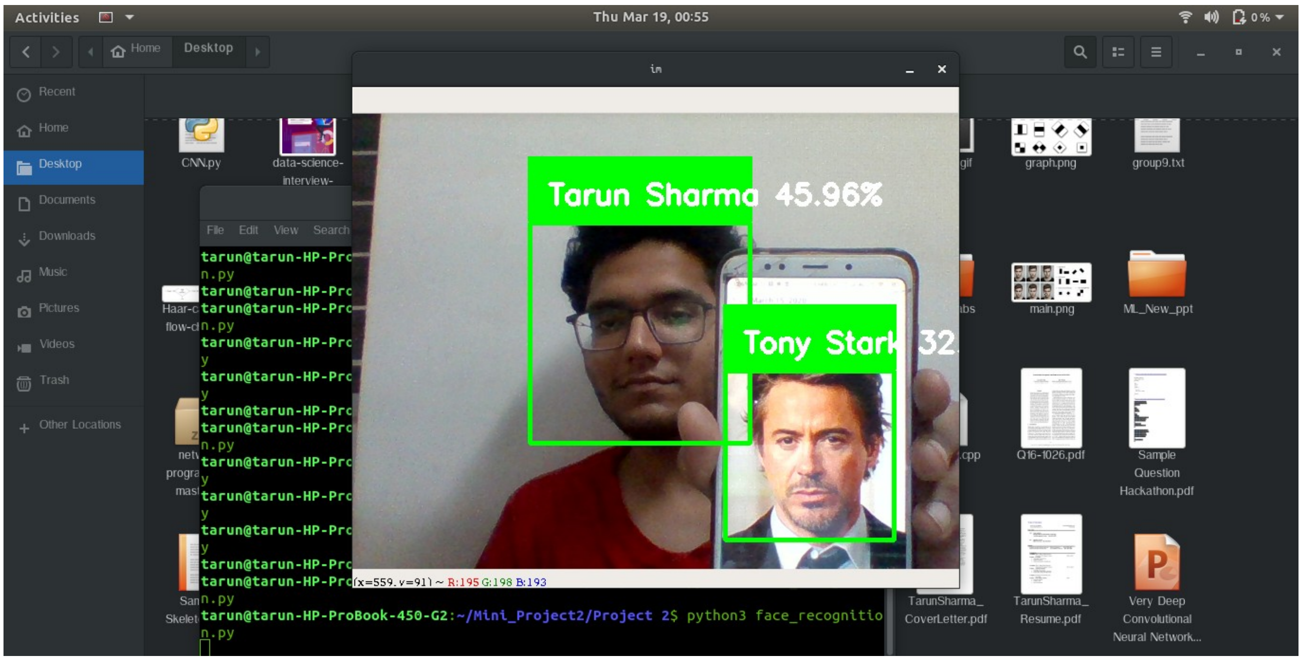
There are total three files named:
1) face_datasets.py
2) face_recognition.py
3) training.py
4) haarcascade_frontalface_default.xml
5) trainer.yml





```
tarun@tarun-HP-Pro
n.py
tarun@tarun-HP-Pro
tarun@tarun-HP-Pro
n.py
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
tarun@tarun-HP-Pro
n.py
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
tarun@tarun-HP-Pro
n.py
tarun@tarun-HP-ProBook-450-G2:~/Mini_Project2/Project 2$ python3 face_recognitio
n.py
```

```
(x=508, y=111) ~ R:200 G:213 B:204
```



```
tarun@tarun-HP-Pro
n.py
tarun@tarun-HP-Pro
tarun@tarun-HP-Pro
n.py
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
tarun@tarun-HP-Pro
n.py
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
y
tarun@tarun-HP-Pro
tarun@tarun-HP-Pro
n.py
tarun@tarun-HP-ProBook-450-G2:~/Mini_Project2/Project 2$ python3 face_recognitio
n.py
```

```
(x=559, y=91) ~ R:195 G:198 B:193
```

4.Future Work And Implementation

I am thinking of making a emotion detection,gender detection,and age detection model using cnn.

5.References

1) https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

2) https://www.researchgate.net/publication/3766402_General_framework_for_object_detection

3) https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf

4)https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html