

**Application of Face Recognition**

by

Talha Tallat

This report is submitted in partial fulfilment of the requirements of the Degree in Electronics and Communications Engineering (DT008) of the Technological University Dublin

April 30th, 2020

Supervisor: Dr Kevin Tiernan

School of Electrical and Electronic Engineering

**DECLARATION**

I, the undersigned, declare that this report is entirely my own written work, except where otherwise accredited, and that it has not been submitted for a degree or other award to any other university or institution.

Signed: \_\_\_\_\_\_\_\_\_\_\_\_Talha Tallat\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_20/04/2020\_\_\_\_\_\_\_\_\_\_\_

**Acknowledgements**

The project would not be possible without the wonderful advice and guidance of a group of remarkable staff members in the school of Electrical and Electronic Engineering at Technological University Dublin.

I would like to thank the project supervisor Dr Kevin Tiernan for his great advice and supervision of the project from the very start till the final stage.

Contents

[1. Introduction 6](#_Toc84119774)

[1.1. Project Overview 6](#_Toc84119775)

[1.1.1. Description of the project report 6](#_Toc84119776)

[1.1.2. Aim of the project report 6](#_Toc84119777)

[2. Project Research 7](#_Toc84119778)

[2.1. Initial Research 7](#_Toc84119779)

[2.2. Face detection 8](#_Toc84119780)

[2.2.1. Anaconda & Python 8](#_Toc84119781)

[2.2.2. OpenCV 9](#_Toc84119782)

[2.2.3. Haar Features 10](#_Toc84119783)

[2.3. Face Recognition 11](#_Toc84119784)

[3. Technical Description & Construction Details 12](#_Toc84119785)

[3.1. Python setup 12](#_Toc84119786)

[3.1.1. Anaconda installation setup 12](#_Toc84119787)

[3.1.2. OpenCV installation setup 13](#_Toc84119788)

[3.1.3. Haars Cascade installation 14](#_Toc84119789)

[3.2. Frontal face detection 14](#_Toc84119790)

[3.2.1. Video stream Face Detection 14](#_Toc84119791)

[3.2.2. Image Face detection 17](#_Toc84119792)

[3.3. Face Recognition 18](#_Toc84119793)

[3.3.1. faceRecognition.py 19](#_Toc84119794)

[3.3.2. teter.py 22](#_Toc84119795)

[4. Test procedures and results 26](#_Toc84119796)

[4.1. Test 1 – Face Detection with an image 26](#_Toc84119797)

[4.2. Test 2 – Image Face Detection 28](#_Toc84119798)

[4.3. Test 3 – Face Recognition with camera 29](#_Toc84119799)

[4.3.1. Tester.py 29](#_Toc84119800)

[4.3.2. FaceRecognition.py 30](#_Toc84119801)

[4.3.3. Results 31](#_Toc84119802)

[4.4. Test 4 - Face Detection with an Image 33](#_Toc84119803)

[5. Conclusion 34](#_Toc84119804)

[6. References 35](#_Toc84119805)

Figures

[Figure 1 - List of Anaconda's available Package [2] 9](#_Toc84119813)

[Figure 2 - Rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles is subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature. [1] 10](#_Toc84119814)

[Figure 3 - Rectangular features in Grayscale image [1] 11](#_Toc84119815)

[Figure 4 - LBP conversion to binary [7] 11](#_Toc84119816)

[Figure 5 - Available versions for Anaconda's installation [2] 12](#_Toc84119817)

[Figure 6 – Installed Python version (3.7.4) 12](#_Toc84119818)

[Figure 7 – Installed Anaconda version (1.7.2) 12](#_Toc84119819)

[Figure 8 – Installed libraries for the test in Console 13](#_Toc84119820)

[Figure 9 - Available OpenCV library to install in Anaconda Navigator 13](#_Toc84119821)

[Figure 10 - Imported libraries in the program 14](#_Toc84119822)

[Figure 11 - Screenshot of the camera working ‘video’ stream 15](#_Toc84119823)

[Figure 12 - Grayscale mode 15](#_Toc84119824)

[Figure 13 - Face detection with a video stream 16](#_Toc84119825)

[Figure 14 - Directory for Face Detection 17](#_Toc84119826)

[Figure 15 - Resized 'photo' window screen to 640x480 [3] 18](#_Toc84119827)

[Figure 16 – Directory 18](#_Toc84119828)

[Figure 17 - Class diagram of Face Recognition 19](file:///F:\COLLEGE%20FILES\2019-2020\Semestor%202\Final%20Year%20Project\Final%20Project%20Report\Final%20Report_Talha%20Tallat.docx#_Toc84119829)

[Figure 18 – This function returns the face part of the grey image along with ID's 19](#_Toc84119830)

[Figure 19 – Data Gathering stored in Database within a directory 19](file:///F:\COLLEGE%20FILES\2019-2020\Semestor%202\Final%20Year%20Project\Final%20Project%20Report\Final%20Report_Talha%20Tallat.docx#_Toc84119831)

[Figure 20 - "Database\_tranningimages" directory ID's for Training images in a database 20](#_Toc84119832)

[Figure 21 - Training images directory for ID 0 20](#_Toc84119833)

[Figure 22 - Training images directory for ID 1 20](#_Toc84119834)

[Figure 23 - This function trains the classifier and takes faces & ID's as returned by the training database function as its argument 21](#_Toc84119835)

[Figure 24 - "testimages" directory for the testing recognizer 22](#_Toc84119836)

[Figure 25 - Fed face data and respective ID's of each face to the recognizer so it can learn 23](file:///F:\COLLEGE%20FILES\2019-2020\Semestor%202\Final%20Year%20Project\Final%20Project%20Report\Final%20Report_Talha%20Tallat.docx#_Toc84119837)

[Figure 26 - Block diagram of the Face Recognition via test image 24](file:///F:\COLLEGE%20FILES\2019-2020\Semestor%202\Final%20Year%20Project\Final%20Project%20Report\Final%20Report_Talha%20Tallat.docx#_Toc84119838)

[Figure 27 - Block Diagram Face Recognition 25](file:///F:\COLLEGE%20FILES\2019-2020\Semestor%202\Final%20Year%20Project\Final%20Project%20Report\Final%20Report_Talha%20Tallat.docx#_Toc84119839)

[Figure 28 - Face Detection program code 26](#_Toc84119840)

[Figure 29 - Face Detection from a Video Camera 27](#_Toc84119841)

[Figure 30 - Detecting face in distance too 27](#_Toc84119842)

[Figure 31 - A working program for the image face detection 28](#_Toc84119843)

[Figure 32 - Result for test 2 28](#_Toc84119844)

[Figure 33 - Face Detection 31](#_Toc84119845)

[Figure 34 - Face Recognition 31](#_Toc84119846)

[Figure 35 - Resized face identification 32](#_Toc84119847)

[Figure 36 - Working code for Face Recognition input from the testing image directory 33](#_Toc84119848)

[Figure 37 - Face Recognition with a provided image 33](#_Toc84119849)

[Figure 38 - New environment created in Anaconda Navigator 34](#_Toc84119850)

[Figure 39 - Command for OpenCV installation in Anaconda prompt 34](#_Toc84119851)

# Introduction

## Project Overview

### Description of the project report

A **Facial recognition** system is a well-known technology capable of identifying or verifying a human face in a digital image, video or a live video source from a camera. This technology detects people’s faces and recognises who the person is. It works by reading the facial features of a given image and compares them with facial features stored within a database. There are two major parts to this project as listed below.

* Face Detection -
* Facial Reignition

This project is suitable for security purposes for recognizing humans and can be used for unlocking doors, surveillance, medical centre, retail stores, etc. For example, this recognizer can work like a home assistant unlocking a door automatically for home users when users are present at the door. If the facial feature of a face matches with a face feature within a database it can provide user access into the house with some interface that can be implemented. However, Face Recognition is the core of identification, so this project mainly focuses on the core alone.

### Aim of the project report

This report contains the research, setup of the face detection and facial recognition of the overall facial reignition project.

The initial aim of the project was to design and program a facial recognition system that can identify human’s faces and making sure that is suitable for security purposes, whether somebody can rely on this system or not.

The project is divided into several different objectives parts that are required to complete this project.

* To read a live video camera for detection & recognition
* To detect a face from a video camera or given image
* Gather and train the data for face identification
* To be able to Recognize a face

The chosen solution requires the use of Anaconda, Python, OpenCV and HaarCascade to be able to process the image or video for recognizing a face. Also, a camera is required to discover & identify the face features over live video. Importantly OpenCV was a huge step to this project since it uses machine learning algorithms to search and identify faces within a picture.

# Project Research

## Initial Research

**Facial Recognition** has made a great impact on the lives of people, thanks to the learning machines that can process face features. The face is a password that can unlock smartphones devices and bank accounts or anything that requires security can be secured. Facial recognition can track any movements, it can guess human’s sexuality through facial features, it can identify people and is capable of storing information about those people. The human face has an amazing variety of features, which does not only help us to recognize others but also allows us to read and understand them through a constant flow of intentional and unintentional signals.

It’s one of the unique functions that separates humans from the machine, until late 2018, where companies like ‘Apple’ launched these technologies that are built into their smartphones, which reads a face to unlock the phone replacing their “touch ID”. A user looks at the phone and it recognises this user in seconds Since there has been a huge amount of progress made in these technologies known as Machine learning, which allows pulling a very accurate faceprint out of a photograph that uniquely identifies a person.

These facial recognition technologies are used every day for things like phone security and serious uses from important organizations, such as banks and governments. Some banks are looking at implementing facial recognition technologies into their ATMs as a security measure for verifying transactions.

Government agencies, such as ICE and the FBI are using facial recognition to create the database from existing documents like drivers’ licenses.

The machine read faces with high precision and performs better when teaching machines more to read faces. This precision makes the technology more effective for any other use. These technologies are used in retail stores to generate data on customers, tracking customers shopping habits and targeting in-store adverts. These facial reignition technologies are even used in churches and to monitor attendance and used in school’s to manage the student’s behaviour and put them into the watch list and these technologies are used in such places as surveillance, medical centre, retail stores, churches, mosques, etc.

The face recognition model can measure the distance between eyes, the width of your lips, the distance from lips to nose, etc. The software has the power to identify one face from millions in the database under one second and these Deep learning methods have accuracy under a controlled environment is around 99%. This technology can identify people without them knowing that they are being identified.

China is already using these technologies widely in security, looking for terrorists, people who have warrants out for their arrest, people can pay in fast-food restaurants, people can access theme parks without having to buy a ticket and also try and catch people who steal toilet papers from the public toilets. Companies in China have access to a government image database of more than 700 million people which is approximately half of its population. These abilities to record, store and analyse images of faces on a vast scale will fundamentally change notions of privacy, fairness and trust.

## Face detection

This Face detection only concentrates on the 2-dimensional frontal face data whereas the deep learning methods are complicated and precise enough, the reason being it process 3-dimensional facial features. The fact that deep learning is used for everything nowadays but in 2002 Paul viola & Michael Jones came up with the following paper, called “Rapid Object Detection using a Boosted Cascade of Simple Features” [1] which these algorithms by Viola/Jones in face detection still performs fine due to its incredibly quick way to calculate features and use them to make a face classification decisions, basically saying is there a face in this block of an image or there isn’t. Most of the camera’s that uses face detection it’s going to be using something very similar to this algorithm. The algorithm is explained in more detail under the ‘Haar Features’ heading below. [1]

The process of face detection and identification is divided into steps to grasp how the project is implemented. Some of the main components and applications were employed at the start of the project and all the procedures for programs setup and detail are as follows.

### Anaconda & Python

**Anaconda** is a free and open-source distribution of the Python and R programming language for scientific computing, such as data science, large scale data processing, machine learning applications etc. Its goal is to provide simplified Python package management and deployment. [2]

Anaconda is very popular because it provides a short and simple setup and most importantly, it has many of the tools used in data science and machine learning with just only one installation. Anaconda Distribution contains Anaconda Navigator, Python and hundreds of scientific packages.

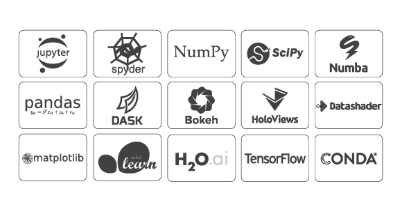


Figure 1 - List of Anaconda's available Package [2]

For this project, Anaconda had to be installed to be able to take advantage of useful components like NumPY and Spyder. Spyder is a MATLAB like IDE for developing python programs and NumPY extends the Python language for numerically intensive scientific application.

Anaconda was used to set up and install OpenCV which is an optimized library for computer vision applications.

### OpenCV

**OpenCV** is an open-source computer vision library originally developed by Intel. It is mostly used for all the various face recognitions, object recognitions, and all other machine learning and deep learning technique. OpenCV is cross-platform itself but originally written in C/C++ and now it provides bindings for various programming languages, such as Python, Java, MATLAB, Interfaces, Windows and Linux.

In this project, OpenCV uses machine learning algorithms to discover and identify faces within a picture. Because the face is a difficult task to perform since it requires thousands of small patterns and features to be matched. These algorithms break the task of identifying the faces into small bite-sized tasks, making it easy to solve. These tasks are also known as classifiers.

The algorithms start at the top left of a picture and move down across small blocks of data, looking at each block, constantly asking, “Is this a face? ... Is this a face? ….” Since there are approximately 6,000 or more tests per block, that might have thousands and millions of calculations which will grind the computer to a halt.

To avoid this, OpenCV uses **cascades**, which breaks the problem of detecting faces into multiple stages. For each block, it performs a rough test, if the test passes then it performs a more detailed test and continues to do for the rest of them. These algorithms may have 30 to 50 of these stages or cascades, and they will only detect a face if all stages pass.

**Cascades** are just a bunch of XML files that contain OpenCV data used to detect objects. Cascade does all the work when initializing code with the selected cascade and these are discussed in depth in the next section.

### Haar Features

The problems with face detection are

* Unknown size of the face - could be big or small
* Very high-resolution image
* Different ethnic groups – young and old people
* People with glasses on

All these problems listed above add up to quite a difficult problem considering that it must be performed lots and lots of times a second and if looking over every tiny bit of image lots and lots of time also considering the trade-off between speed and accuracy and false-positives and false-negatives is a quite slow and complicated process. Which makes it difficult to find a face quickly.

Paul Viola & Michael Jones came up with a classifier that uses very simple features, one bit of an image subtracted by from another bit of an image in thousands and thousands of those features providing a clue after a good few stages, maybe there is a face. It is making very quick decisions about what it is to be a face.

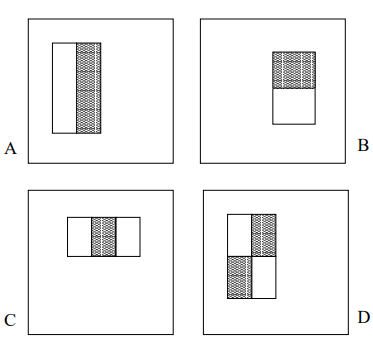


Figure 2 - Rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles is subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature. [1]

Looking at the grayscale image in the figure shows the eye is slightly darker than the forehead & nose in terms of shadowing and the pupils are even darker. So, white rectangle minus the darker rectangle over an eye and mostly, an eye is going to have a different response from a white background in the figure. It can get a lot of the faces but also it finds a load of other stuff too, where something happens to be darker than something else, that happens all the time.

Violas and Jones produce a lot of these features working all at ones and deciding this way.



Figure 3 - Rectangular features in Grayscale image [1]

When the image is small this can have a lot of different possible features even of these four types shown in figure (). These four rectangle features can be one pixel each or one of them can be half the image, meaning that it can scale and move around.

## Face Recognition

All the features from the Face Detection had to be used since the detection is only a major step for this Face Recognition project. For face Recognition an LBPH recognizer must be used

**LBPH Face Recognizer Process**

Considering a 3×3 window and move it across one image. At each move (each local part of the picture), compare the pixel at the centre, with its surrounding pixels. Denote the neighbours with intensity value less than or equal to the centre pixel by 1 and the rest by 0. [7]

After reading these 0/1 values under the 3×3 window in clockwise order, you will have a binary pattern like 11100011 that is local to an area of the picture. When you finish doing this on the whole image, you will have a list of **local binary patterns**. [7]

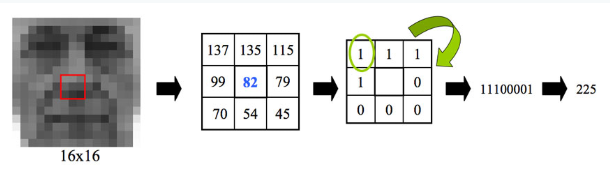


Figure 4 - LBP conversion to binary [7]

# Technical Description & Construction Details

## Python setup

A Python programming language was chosen to start this face recognition project to be able to use the open-source library OpenCV for image processing. A well-known “**Anaconda**” free and open-source distribution of Python had to be installed to run Python on Windows operating system.

### Anaconda installation setup

Anaconda was installed on the windows operating system which contains the latest version of Python along with other useful packages shown in figure 5. The download link for the installation was available on Anaconda’s original website. [2]



Figure 5 - Available versions for Anaconda's installation [2]

The latest version of Python 3.7 (64-Bit) was installed on the computer with a size of 462 MB, however, there were “Python 3.7” and Python 2.7” versions available on the Anaconda official website for windows installer as shown in figure ().

The following commands “python -V” & “Anaconda -V” were used in Anaconda Prompt to check the right versions are installed for the Python and Anaconda as shown in the figures ( - ).

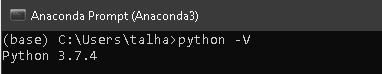


Figure 6 – Installed Python version (3.7.4)

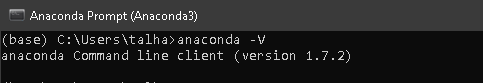


Figure 7 – Installed Anaconda version (1.7.2)

The beauty of Anaconda is that it has most of the required libraries needed for the project such as “sys” and “numpy as np”. These were imported in the Python console to test the libraries, whether they are installed or not. If not, an error will show up, meaning that the libraries are not installed otherwise there is no error.



Figure 8 – Installed libraries for the test in Console

The most important library that was installed from Anaconda Navigator is OpenCV “cv2”. OpenCV library was installed properly without an error since it was checked in the console window by importing “import cv2” in the Console window as shown in figure ().

### OpenCV installation setup

OpenCV was a convenient way to detect a face using Python language via Anaconda Navigator and it was the most appropriate way to install OpenCV.

OpenCV is a library for image processing and Computer Vision which is necessary for face recognition as discussed in more detail in chapter 2.

The Anaconda Navigator looked for the OpenCV when it was searched on the Anaconda Navigator search bar and all available OpenCV libraries options were shown and there were three main libraries available for OpenCV to install “libopencv”, “opencv” and “py-opencv” as shown in figure (). The “opencv” library was installed and It didn’t take much time to install as compared to manually installed. The manual installation of OpenCV had flaws which are discussed in more detail in chapter 5.

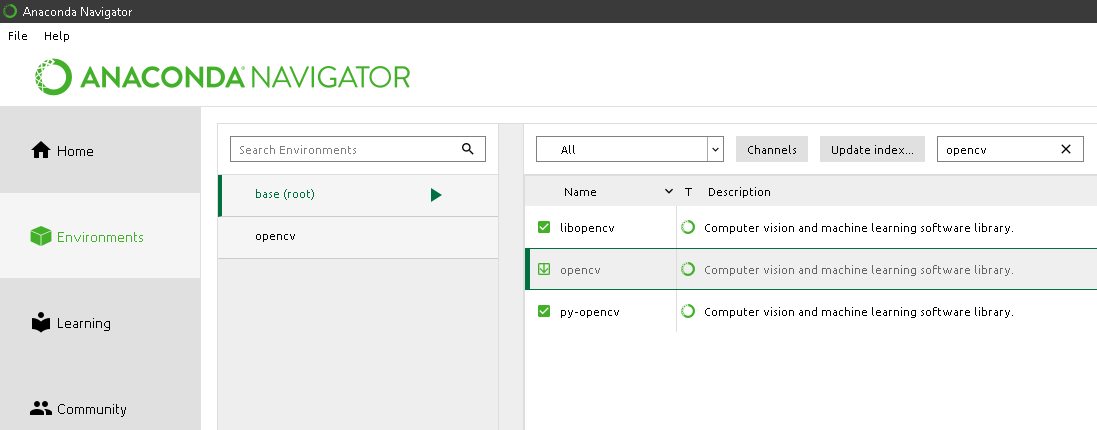


Figure 9 - Available OpenCV library to install in Anaconda Navigator

The OpenCV was imported in the Python console in ‘Spyder’ to check whether the “OpenCV” library was installed or not and OpenCV was installed properly without an error as shown in figure ().

The ‘Spyder’ is an application package that runs a python programming language which comes with Anaconda and that is where all the python program was written.

### Haars Cascade installation

The Haars cascade is an XML file also known as a classifier which was required for frontal face detection and the OpenCV has many pre-trained classifiers for face, eyes, smile etc. This face cascade was the most common way to detect a face and it is available on the OpenCV GitHub page. [2]

This “haarcascade\_frontalface\_default.xml” frontal face cascade was extracted after the installation and added in the same directory where the program is saved, and this provides an easy location path when using it in the program.

## Frontal face detection

Face detection is the most basic task for Face Recognition. The two main tasks were to write a program that detects a face by providing an image from the directory and detect a face from the captured video camera.

The OpenCV virtual environment was completely ready to run since all the packages were installed so, the new project was created in the “Spyder” Python 3.7 version and the name of the program file “faceDetection” was saved in the directory.

### Video stream Face Detection

The purpose was to capture a live face video with the help of a camera for live Face Detection. The camera was tested before starting to program for live face detection.

To start the program the “numpy as np” and “cv2” libraries were imported first.

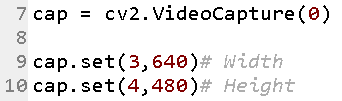
NumPy converts python lists to NumPy arrays as OpenCV face recognizer needs arrays for recognition.



Figure 10 - Imported libraries in the program

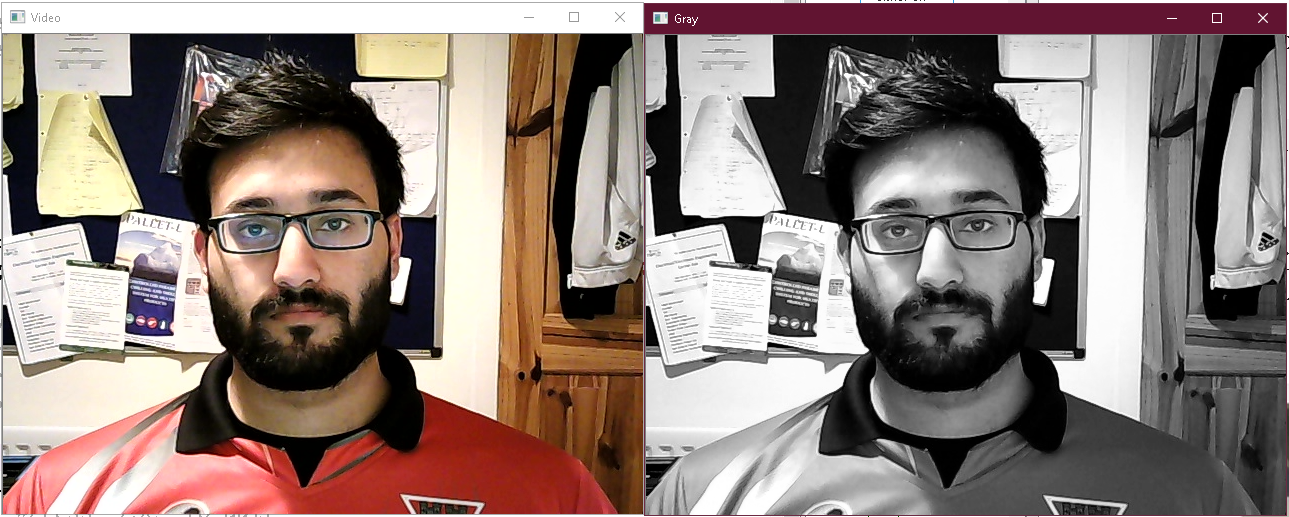


The 5th line loads the “classifier” that is stored in the directory.



The code in line 7 captures the video stream generated by the camera and the (0) represent the camera number one available.

The 9th line sets the width to 640 and the 10th line sets the height to 480 which is the captured window screen from a camera and the window screen width and height is 640x480 shown in figure (). The size of the window can vary depending on the choice.



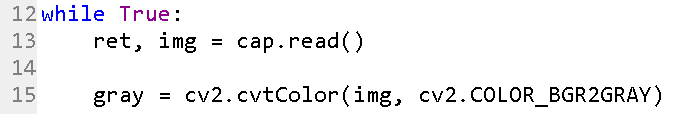
**480**

**(0,0)**

**640**

**(640, 480)**

Figure 11 - Screenshot of the camera working ‘video’ stream



The 13th line set the camera and line 15th loads the input video in grayscale mode inside the loop. Grayscale is discussed in more detail in chapter 2.

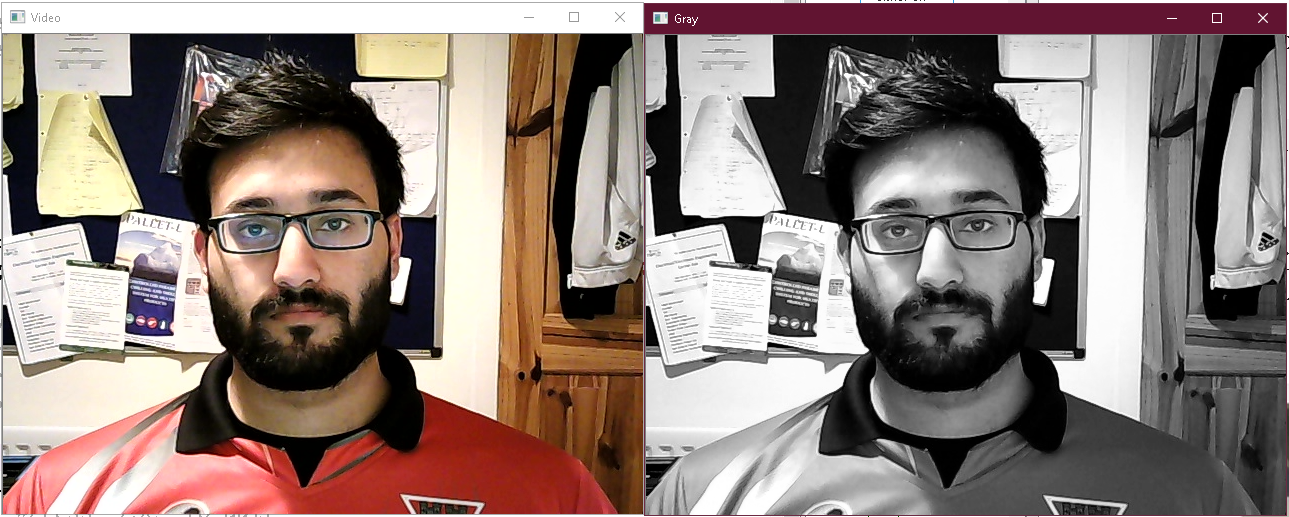


Figure 12 - Grayscale mode

The most important passing parameters for the classifier function were scale factor, number of neighbours and minimum size of the detected face. The function below detects faces on the image.

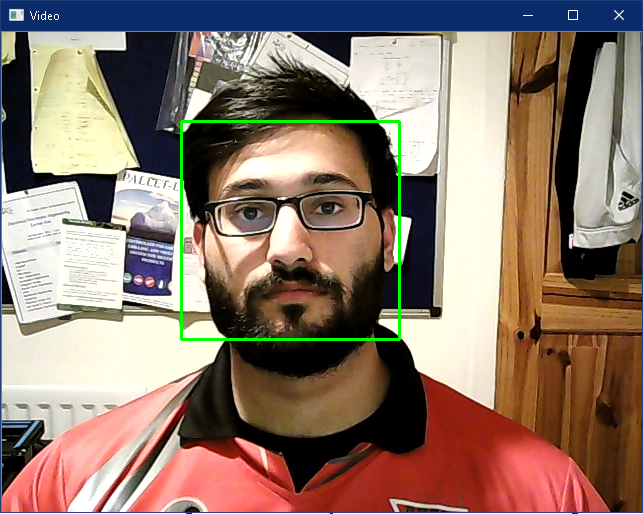


* **‘gray’** is the input grayscale image shown in figure ().
* **‘scaleFactor’** is used for creating the scale pyramid and this parameter identifying how much the size of the image is reduced at each image scale.
* **‘minNeighbbors’** is a parameter that identifies how many neighbours each applicant rectangle should have to keep it. A higher number provides lower false positives.
* **‘minSise’** is the minimum rectangle size to be considered a face.



The line from 20 – 23 is responsible for marking green squares around the face. If a face is found, then it returns the positions of a detected face like a square, clearly shown (x,y,w,h) in figure ().

The green colour is dependent on these perimeters (0,255,0), which represents the colour intensity varying from 0-255 and the order is in Blue, green, and red (BGR) makes the green colour bright at 100% intensity and the rest of the colours are at 0% intensity, means turned off.



**(x, y)**

**h**

**W**

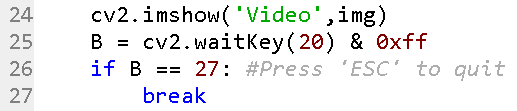
**480**

**(0,0)**

**(640, 480)**

**640**

Figure 13 - Face detection with a video stream



Line 24 shows the ‘Video’ window captured image and if Esc is pressed on the keyboard while running the program then the ‘Video’ window shut off which makes it easy to run again.



If the Esc key is pressed then the ‘Video’ window shuts off, which means it clears the running window.

### Image Face detection

The purpose was to detect a face when providing an image rather than from the live video stream. The code was very similar to the previous code, live video stream face detection.

So, a couple of changes were made to the program.



The 6th line of code reads the path of an image “Photo2.jpg” stored in the “Photos” folder inside the directory as shown in figure () and the 1 represents the colour photograph, whereas before the photograph was taking from the live stream camera.

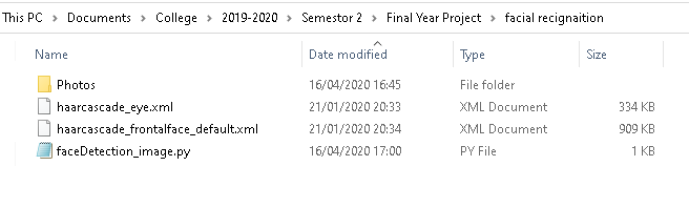


Figure 14 - Directory for Face Detection



The original ‘photo’ window size was too big so the 7th line resizes the image to 640x480 window and the photo path is from the directory and so line 8 prints out the resized “photo” window shown in the figure ().

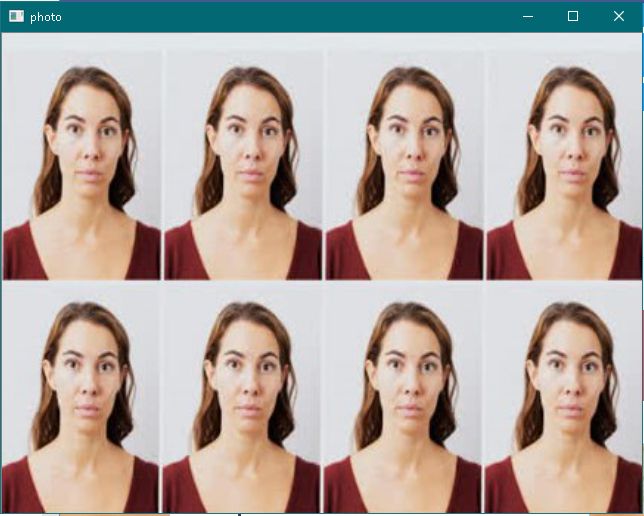
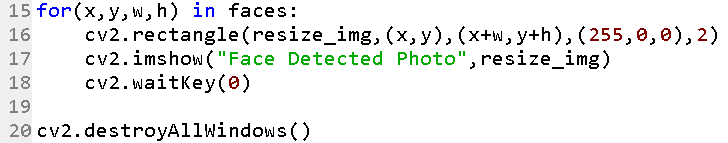


Figure 15 - Resized 'photo' window screen to 640x480 [3]



The line above performs a similar action as before the only thing is that line 17 prints out the resized photo window “Face detected Photo” with the square around the faces because of line 16 as discussed before. The next line 18 holds the running program in memory until Esc is entered.

The results are shown in the chapter 4 result section in more detail.

## Face Recognition

Face Recognition uses all the features from Face detection and all the setups shown above are necessary for Face recognition. The directory was required for creating a database where all the training images were stored, testing images samples as an input for reading the images or capturing video from a camera and frontal face cascade. The program had to be stored in the same directory providing easy access to a database, besides 2 python scripts that were created for the Face Recognition.

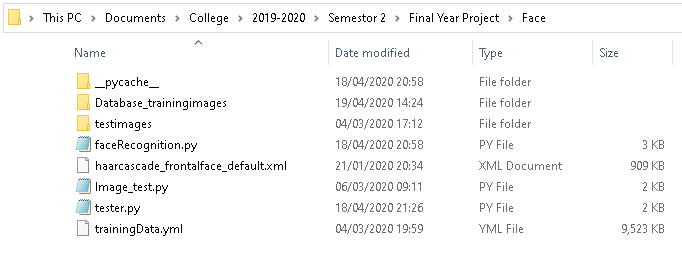


Figure 16 – Directory

The code was very similar to the code that was used for Face Detection and only a couple of function was added on to the face recognition program as follows.

The “faceRecognition.py” contains 5 functions in this program code and “tester.py” is the main code that is inherited from “faceRecognition”.

Figure 17 - Class diagram of Face Recognition

faceDetection

ID\_for\_tranning\_database

train\_classifier

draw\_rect

put\_text

tester.py

faceRecognition.py

Main

### faceRecognition.py

The ‘faceDetection’ function is already was performed in the Face Detection.



Figure 18 – This function returns the face part of the grey image along with ID's

This function had to be created to generate an ID for each of the images in the training database. The Haar classifier accepts images and the ID’s associated with them and the ID’s should only be integers numbers. The structure of the database is shown below in more detail.



**ID = 0**

**ID = 1**

**Data base**

**Gathering face data images for identification**



Figure 19 – Data Gathering stored in Database within a directory

100 training images were taken from the camera for user ID = 0 and 40 training images for user ID = 1 were taken from Google images for only testing purposes.

The ID 0 and ID 1 directory was created as shown in figure () which stores training images samples for the recognizer.

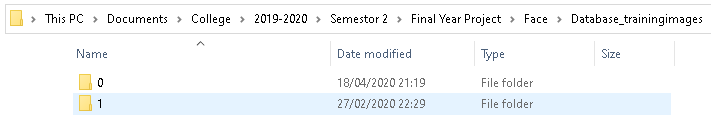


Figure 20 - "Database\_tranningimages" directory ID's for Training images in a database

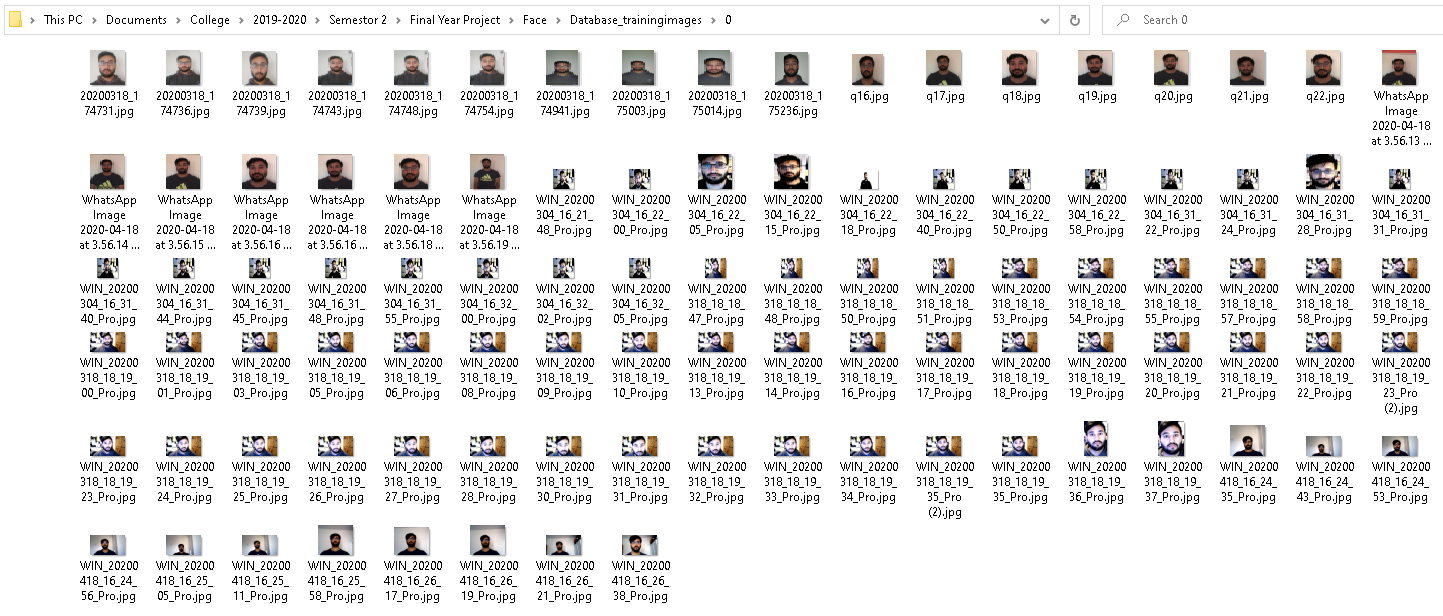


Figure 21 - Training images directory for ID 0

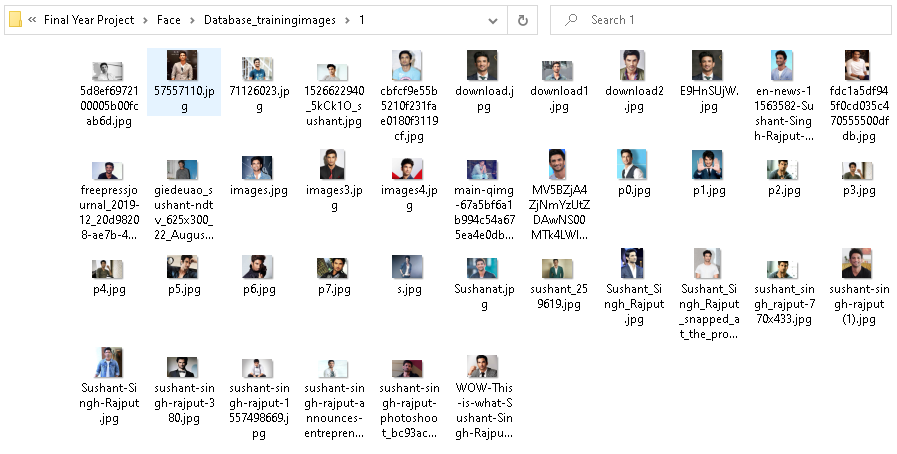


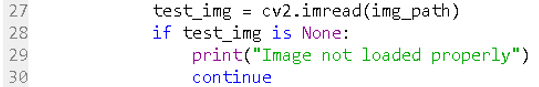
Figure 22 - Training images directory for ID 1

After the directory and the training, the database was set up, the ‘**os.walk**’ had to be used in line from 20 to 24 that allows fitting in a directory to provide a path of the subdirectory such as (ID 0, ID 1) and file names.

**‘os.walk’** goes recursively into directories so it starts from going to database directory where it access folder 0 and folder 1 it goes throw each folder. It goes throw directories within individual directories.



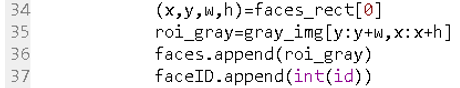
Line 25 and 26 prints out the filename for debug purposes.



Line 27 loads the image one by one from an ‘img\_path’ and if the image does not get loaded properly inline 28 it might halt so then line 29 prints out the “Image not loaded properly” and then line 30 just continues the rest of the process.



Line 31 calls out the ‘faceDetection’ function which return faces detected in the image and considering only a single image is fed to a classifier. The image in the ID 0 folder only contains photographs of the 1st person and a photograph of the 2nd person is stored in folder 1 as shown in figure 18 & 19 because the classifier gets confused if there are 2 different photographs of different people in the same folder. So, in case if the images have multiple faces, more than one then line 32 faces are not equal to one and line 33 skips it.



Inline 34, the rectangular (x,y,w,h) is returned by the faces\_rect module and the entry is 0. Line 35 extracts the face part of the image and since the classifier only takes the grey image, so this cropped region (x,y,w,h) part is taken from the grey image is then feed to (x,y,w,h) part of the classifier.

Line 36 takes in the resized grey image and inline 37, Classifier only takes ID’s in an integer as discussed before so line 37 converts this into “int” ID.



Line 38 returns faces and their ID’s.

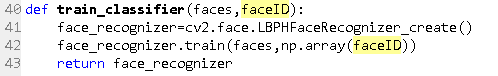


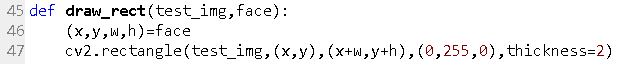
Figure 23 - This function trains the classifier and takes faces & ID's as returned by the training database function as its argument

Defining another function inline 40 to train the classifier on training images ‘faces,faceID’ from the database where ‘faces’ and ‘faceID’ returns grey image inline 38 by the previous function ‘ID\_for\_training\_database’. It provides the faces and the part of the face, where inline 36, it crops that part of the face from an image and the part of an image which is the face of a grey image.

Inline 41, **LBPH** **(Local Binary Patterns Histograms)** face recognizer was created in the program “LBPHFaceRecognizer\_create()” and was loaded onto the variable “face\_recognizer”. **LBPH** analyses each face in the database separately and independently as discussed in chapter 2.

Generated data were trained and this “LBPH” recognizer takes the ID’s as ‘numpy’ arrays, so line 42 creates face ID’s “faceID” array into a ‘numpy’ array.

Line 43 returns the data for the face\_recognizer.



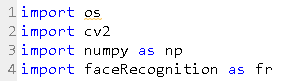
The function inline 45 draws a box around the detected face in the image. ‘detectMultiscale’ returns a rectangle coordinate of the face detected in the test\_img. Line 46 extracts all the coordinates and line 47 calls out the rectangle function from the OpenCV, colour and thickness.



This function in lines 50 and 51 writes the name of the person for detected ID’s

### teter.py

The “faceRecogination” program file was imported to the “tester.py” to use “faceRecognition” modules and the rest of the useful libraries were imported too.





Line 6 loads “Photo5.jpg” image from the “testimages” directory. Line 7 variables call out the ‘faceDetection' function in the ‘faceRecgnition.py’ and line 8 prints out the data in ‘face\_detected’. Varying the size of the image for varying these parameters “scalefactor”, “minNeighbour” and “minSize”.

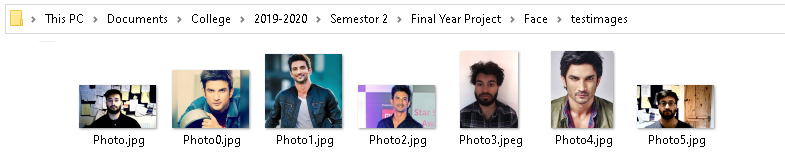


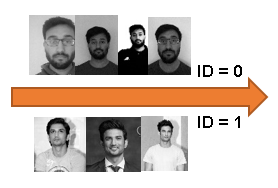
Figure 24 - "testimages" directory for the testing recognizer



Line 10 reads this “ID\_for\_tranning\_database” function from ‘faceRecognizer.py’ which provides the path to the ‘database\_trainingimages’ directory. Line 11 goes back to the “train\_classifier” function which trains haar classifier and takes faces and faced returned by “ID\_for\_tranning\_database” function.

Line 12 takes all the data from the database and writes to the “trainingData” the OpenCV Recognizer. The result is saved in tranningData.yml file as it is done by the specific OpenCV function.

Trainer.yml



Grey images



Figure 25 - Fed face data and respective ID's of each face to the recognizer so it can learn

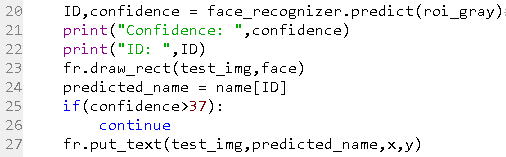
**Recognizer**



Line 16 shows 3 dictionaries were created containing names for each ID stored in the “Data\_tranningiamges” folder within a directory.



In a group photograph, all those ‘faces\_detected’ are taken the first ‘face’ and taking its corresponding (x,y,w,h) coordinate and ‘gray\_image’ was returned by the faces\_detected from line 7 and extracting the resign of interest which is the ‘face’ and then this is fed to a classifier.



The function inline 20 “Predict” is from OpenCV and gave this “roi\_gray” image and the part of the image is a face and it returns an ID (0 or 1) and it also returns a confidence value. Line 21 and 22 prints the Confidence and ID in the console for debugging. Line 23 takes the image from the ‘testimages’ folder and it draws the rectangle around the image and line 24 extracts the ID name out of the ID.

C**onfidence** of 0 represents a perfect match where 100 is a poor match, the threshold value for the confidence is 37, if the confidence level is above 37, it is not required to predict that value because that it is the wrong face. However, it predicts all the values with some confidence where it is not confident enough.

So, if the confidence is below 37 it puts the name of the ID that the face belongs to.

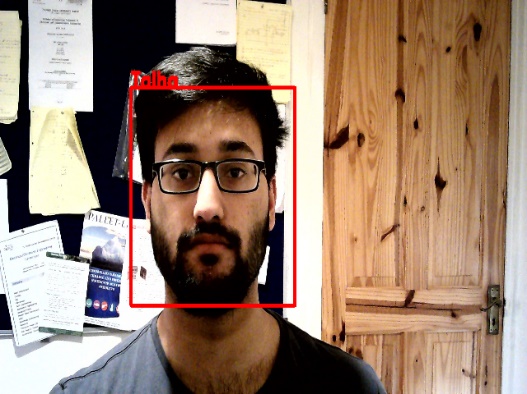


Line 29 resizes the images because it does not fit in the screen properly and line 30 shows the face Recognition.



Line 31 waits indefinitely after the program is run and the Esc key destroys all windows.

**Talha**







**Testing images**

**Confidence = 33**

**ID = 0**



ID = 0

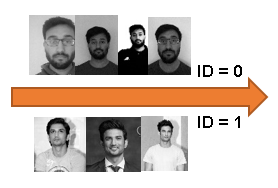
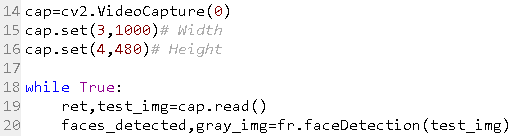




Figure 26 - Block diagram of the Face Recognition via test image

The process of face-recognizing with a live camera is as follows.



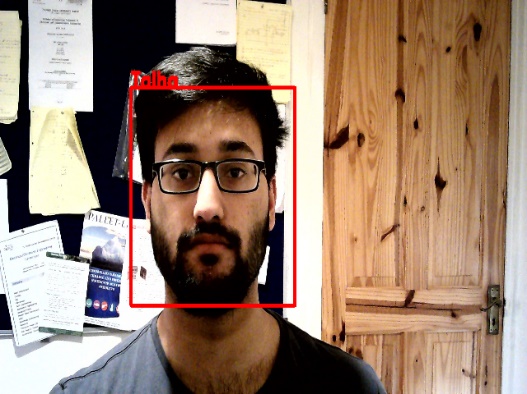
The code above captures the video stream generated by the camera and the (0) represent the camera number one available.

The nest two lines set the width to 1000 and the height to 480 which is the captured window screen from a camera.

Then the ‘cap’ is used as an input live image.



**Talha**







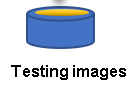
**Confidence = 33**

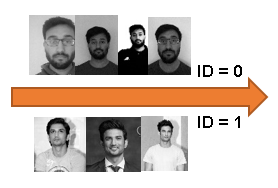
**ID = 0**

**Confidence = 23**

**ID = 2**











ID = 0



Figure 27 - Block Diagram Face Recognition

# Test procedures and results

## Test 1 – Face Detection with an image

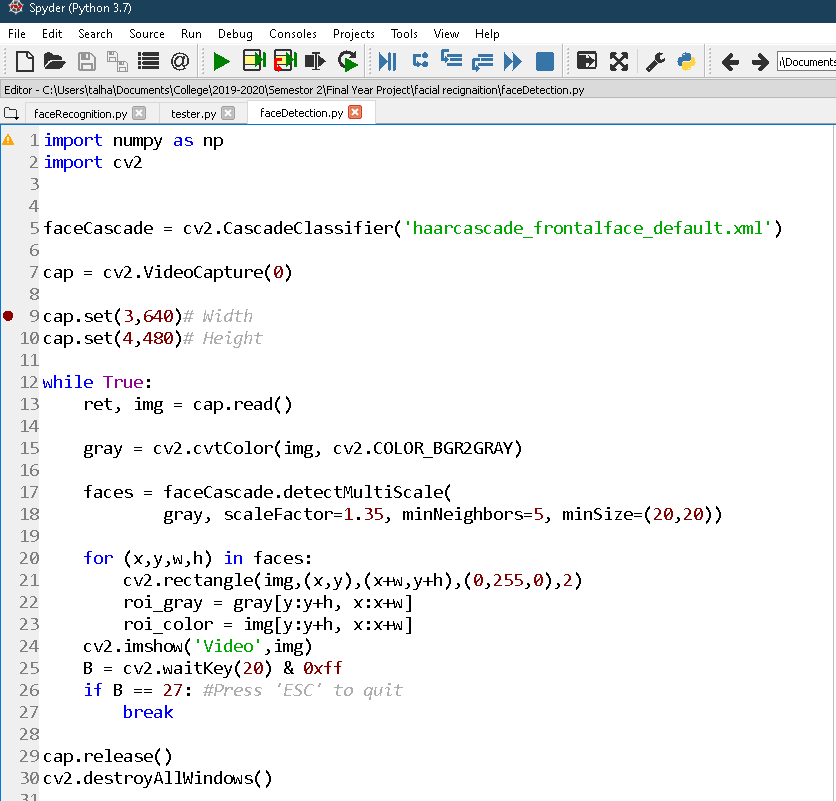


Figure 28 - Face Detection program code

The face detection from a captured video stream was performed in under 25 lines of python code and the result are remarkably shown below as it was detecting the face extremely well even though there are a lot of objects happen to be darker than other at the background.

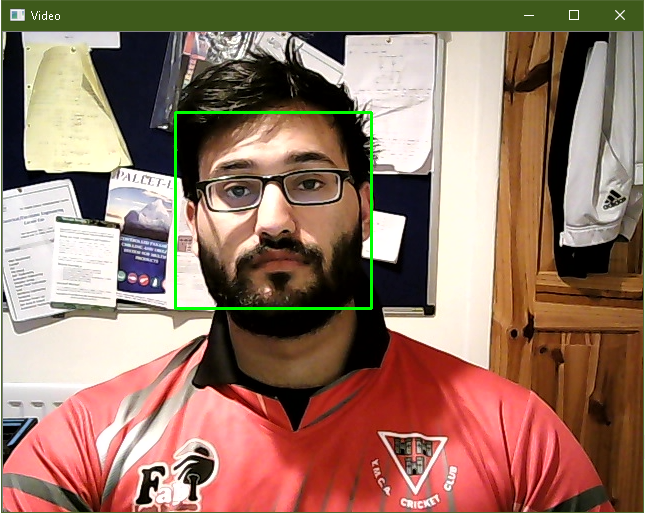


Figure 29 - Face Detection from a Video Camera

The program worked flawlessly without any false detection even though there were so many objects in the background.

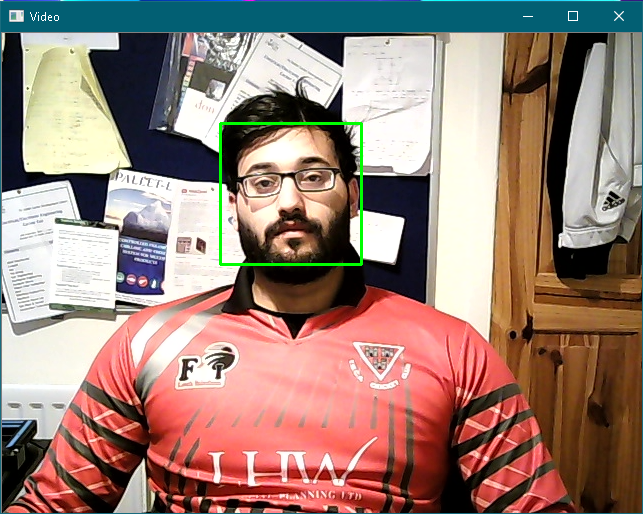


Figure 30 - Detecting face in distance too

The size of the live video can vary for Detection since it works for smaller and larger faces too with the scale factor set to 1.35 and minNeighbour is 5 and the minSize is 30,30.

This is described in more detail for the next test.

## Test 2 – Image Face Detection

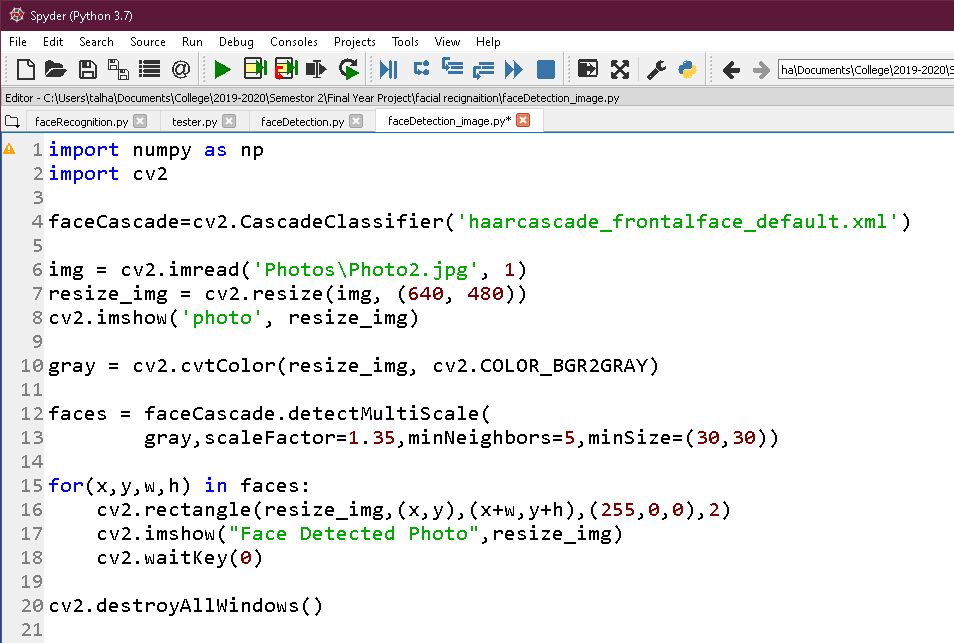


Figure 31 - A working program for the image face detection

The face detection from a provided image was performed in under 25 lines of python code and the result are remarkably shown below as it was detecting 8 faces well.

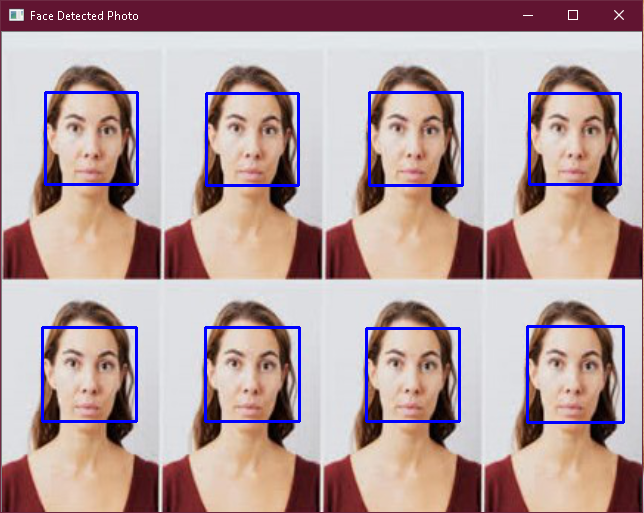


Figure 32 - Result for test 2

The result for face detection from an image is shown above.

## Test 3 – Face Recognition with camera

### Tester.py

import os

import cv2

import numpy as np

import faceRecognition as fr

#This module captures images via webcam and performs face recognition

face\_recognizer = cv2.face.LBPHFaceRecognizer\_create()

face\_recognizer.read('trainingData.yml')#Load saved training data

name = {0 : "Talha",1 : "Kangana", 2 : "Usama"}

cap=cv2.VideoCapture(0)

cap.set(3,1000)# Width

cap.set(4,480)# Height

while True:

ret,test\_img=cap.read()# captures frame and returns boolean value and captured image

faces\_detected,gray\_img=fr.faceDetection(test\_img)

for (x,y,w,h) in faces\_detected:

cv2.rectangle(test\_img,(x,y),(x+w,y+h),(0,0,255),thickness=2)

resized\_img = cv2.resize(test\_img, (640, 480))

cv2.imshow('Face Detection',resized\_img)

cv2.waitKey(10)

for face in faces\_detected:

(x,y,w,h) = face

roi\_gray = gray\_img[y:y+w, x:x+h]

ID,confidence = face\_recognizer.predict(roi\_gray)#predicting the id of given image

print("Confidence: ",confidence)

print("ID: ",ID)

fr.draw\_rect(test\_img, face)

predicted\_name = name[ID]

#cv2.putText(img, str(confidence),(x+5,y+h-5), font, 1, (255,255,0),1)

#fr.put\_confidance(test\_img,confidence,x,y)

if confidence < 40:#If confidence less than 37 then don't print predicted face text on screen

fr.put\_text(test\_img,predicted\_name,x,y)

resized\_img = cv2.resize(test\_img, (640, 480))

cv2.imshow('Face Recognition',resized\_img)

if cv2.waitKey(10) == ord('b'):#wait until 'b' key is pressed

break

cap.release()

cv2.destroyAllWindows

### FaceRecognition.py

import cv2

import os

import numpy as np

def faceDetection(test\_img):

gray\_img=cv2.cvtColor(test\_img,cv2.COLOR\_BGR2GRAY)

face\_haar\_cascade=cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

faces=face\_haar\_cascade.detectMultiScale(gray\_img,scaleFactor=1.32,minNeighbors=5,minSize=(30,30),flags=0)

return faces,gray\_img

def ID\_for\_training\_database(directory):

faces=[]

faceID=[]

for path,subdirnames,filenames in os.walk(directory):

for filename in filenames:

id=os.path.basename(path)

img\_path = os.path.join(path, filename)

print("img\_path:",img\_path)

print("id:",id)

test\_img = cv2.imread(img\_path)

if test\_img is None:

print("Image not loaded properly")

continue

faces\_rect,gray\_img = faceDetection(test\_img)

if len(faces\_rect)!=1:

continue

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

roi\_gray=gray\_img[y:y+w,x:x+h]

faces.append(roi\_gray)

faceID.append(int(id))

return faces,faceID

def train\_classifier(faces,faceID):

face\_recognizer=cv2.face.LBPHFaceRecognizer\_create()

face\_recognizer.train(faces,np.array(faceID))

return face\_recognizer

def draw\_rect(test\_img,face):

(x,y,w,h)=face

cv2.rectangle(test\_img,(x,y),(x+w,y+h),(0,0,255),thickness=10)

def put\_text(test\_img,text,x,y):

cv2.putText(test\_img,text,(x,y),cv2.FONT\_HERSHEY\_DUPLEX,2,(0,0,255),thickness=10)

### Results

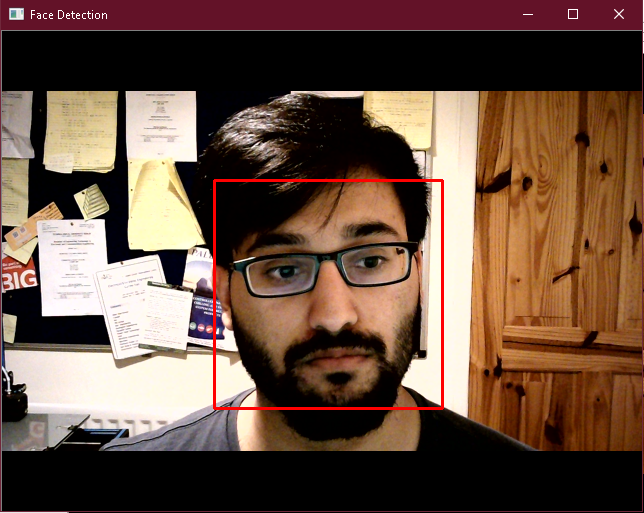


Figure 33 - Face Detection

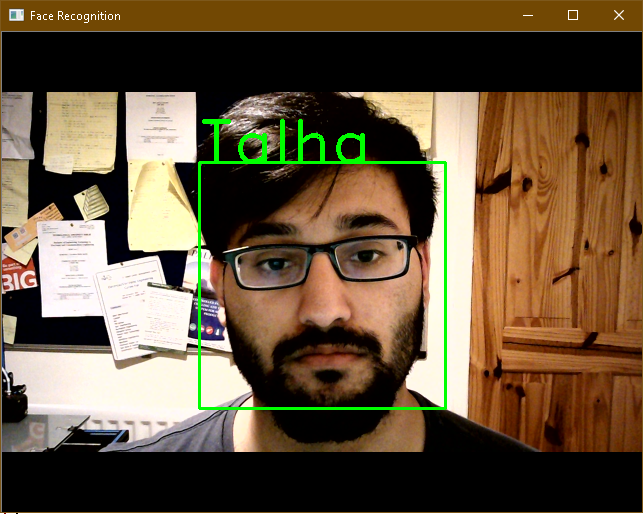


Figure 34 - Face Recognition

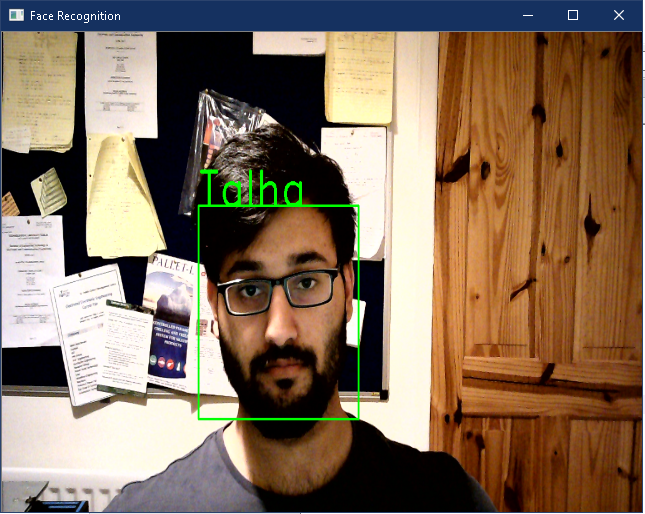


Figure 35 - Resized face identification

## Test 4 - Face Detection with an Image



Figure 36 - Working code for Face Recognition input from the testing image directory

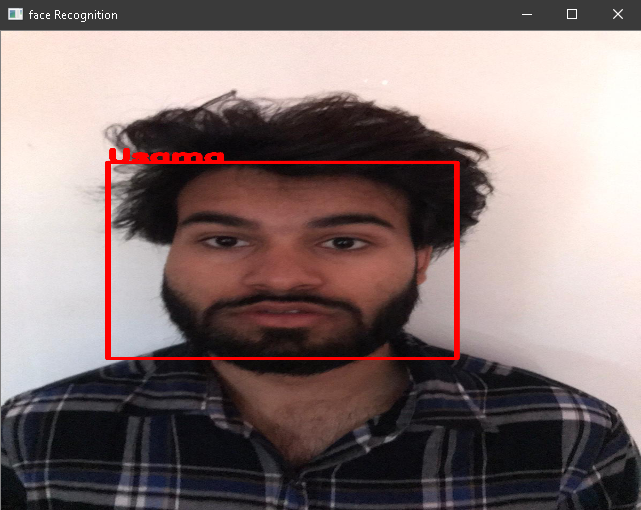


Figure 37 - Face Recognition with a provided image

# Conclusion

The program works considerable well since it can detect and recognise faces and can be improved even more by training hundreds and thousands of training images in the database more. This will increase the processing time for training images but it gives an accurate result with smaller confidence vale.

The most difficult task had to install OpenCV where it took nearly weeks to figure out the right solution to install OpenCV when a new environment had to be created in Anaconda Navigator to install OpenCV packages with an older version of Python 3.6. Since the latest 3.7 version was not supporting. Then OpenCV had to be activated in the Anaconda Prompt, which was creating a lot of issues.

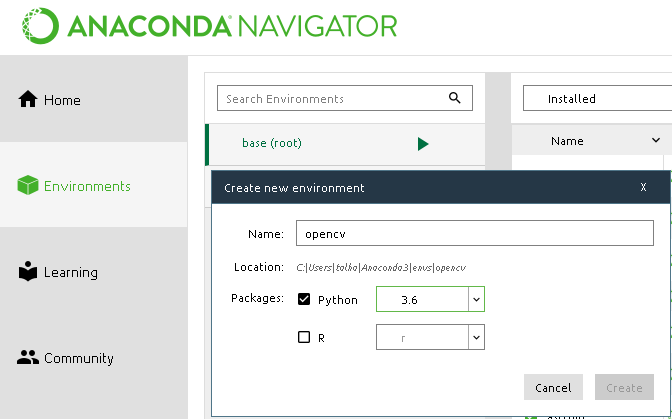


Figure 38 - New environment created in Anaconda Navigator

The following command is typed in the Anaconda prompt to install the OpenCV.

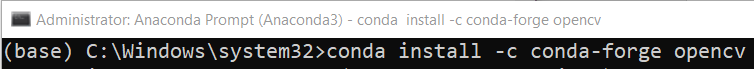


Figure 39 - Command for OpenCV installation in Anaconda prompt

This process took too long to install those packages due to the internet speed and it was not resolving the installation problems, because installing OpenCV manually was a difficult task since it was taken one and half days. A simple way was to install OpenCV was through Anaconda Navigator, which provides with OpenCV option to install it.

# References

[1] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", *Cs.cmu.edu*, 2001. [Online]. Available: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf. [Accessed: 12- Mar- 2020].

[2] "Anaconda Python/R Distribution - Free Download", *Anaconda*, 2020. [Online]. Available: https://www.anaconda.com/distribution/. [Accessed: 20- Feb- 2020].

[3] A. Golubev and D. Kurtaev, "opencv/opencv", *GitHub*, 2020. [Online]. Available: https://github.com/opencv/opencv/tree/master/data/haarcascades. [Accessed: 20- Jan- 2020].

[4] "Passport Photos", *Photobox.ie*, 2020. [Online]. Available: https://www.photobox.ie/shop/prints/passport-id-photos. [Accessed: 25- Feb- 2020].

[5] P. Joshi, "Installing OpenCV 3 With Python On Mac OS X", *PERPETUAL ENIGMA*, 2020. [Online]. Available: https://prateekvjoshi.com/2015/10/09/installing-opencv-3-with-python-on-mac-os-x/. [Accessed: 20- Apr- 2020].

[6] "Noon - Unsupported Browser", *Noon.com*, 2020. [Online]. Available: https://www.noon.com/uae-en/c6035-wired-web-cam-black/N21003158A/p?o=dd9c0316c5e9907d. [Accessed: 20- Apr- 2020].

[7] R. Raja, "SuperDataScience", *Superdatascience.com*, 2020. [Online]. Available: https://www.superdatascience.com/blogs/opencv-face-recognition. [Accessed: 09- Feb- 2020].