**Computer Engineering Department**

**Faculty of Engineering**

**Islamic University of Gaza**

**Palestine, Gaza**

# Face Recognition based attendance system

**Final Graduation Project Report**

**Submitted in Partial Fulfillment of the Requirements**

**For**

**The Bachelor of Computer Engineering Degree**

Prepared by

Dalia Taysir Younis    Eman Adel Ibrahim

Iman Issa Siam    Nesma Khalil El-Safadi

Professor

Aiman Ahmed Abu Samra

Jan,2021

## Acknowledgements

We wish to thank various people for their contribution to this project. We would like to express our deep gratitude to Professor Aiman Ahmed Abu Samra, our research supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. We also would  like to thank Eng. Aziza for her useful and constructive recommendations on this project. Finally, we wish to thank our parents for their support and encouragement throughout our study.

# DEDICATION

We dedicate our work to our families and our friends. A special feeling of gratitude to our loving parents. In addition, for all teachers of the faculty computer engineering that teach us to be passion for learning and practice our experiments.

# Abstract

In the 21st century, everything around us has become depends upon technology to make our life much easier. Daily tasks are continuously becoming computerized. Nowadays more people prefer to do their work electronically. To the best of our knowledge, the process of recording students' attendance at the university is still manual. Lecturers go through manual attendance sheets and signed papers to record attendance. This is slow, inefficient and time consuming. The main objective of this project is to offer system that simplify and automate the process of recording and tracking students' attendance through face recognition technology. It is biometric technology to identify or verify a person from a digital image or surveillance video. Face recognition is widely used nowadays in different areas such as universities, banks, airports, and offices. We will use preprocessing techniques to detect, recognize and verify the captured faces like Eigenfaces method. We aim to provide a system that will make the attendance process faster and more precisely. The core problem is identified along with solutions and project path. Furthermore, detailed system analysis and design, user interface, methods and the estimated results are presented through our documentation.

# Introduction

Every organization requires a robust and stable system to record the attendance of their students. and every organization have their own method to do so, some are taking attendance manually with a sheet of paper by calling their names during lecture hours and some have adopted biometrics system such as fingerprint, RFID card reader, Iris system to mark the attendance. The conventional method of calling the names of students manually is time consuming event. The RFID card system, each student assigns a card with their corresponding identity but there is chance of card loss or unauthorized person may misuse the card for fake attendance. While in other biometrics such as finger print, Iris or voice recognition, they all have their own flaws and also, they are not 100% accurate. Use of face recognition for the purpose of attendance marking is the smart way of attendance management system. Face recognition is more accurate and faster technique among other techniques and reduces chance of proxy attendance. Face recognition provide passive identification that is a person which is to be identified does not to need to take any action for its identity. Face recognition involves two steps, first step involves the detection of faces and second step consist of identification of those detected face images with the existing database. There are number of face detection and recognition methods introduced. Face recognition works either in form of appearance based which covers the features of whole face or feature based which covers the geometric feature like eyes, nose, eye brows, and cheeks to recognize the

face. Our system uses face recognition approach to reduce the flaws of existing system with the help of machine learning, it requires a good quality camera to capture the images of students.

## Objectives

- Detection of unique face image amidst the other natural components such as walls, backgrounds etc.
- Extraction of unique characteristic features of a face useful for face recognition.
- Detection of faces an amongst other face characters such as beard, spectacles etc.
- Effective recognition of unique faces in a crowd (individual recognition in the crowd)
- Automated update in the database without human intervention.
- Provides a valuable attendance service for both teachers and students.
- Reduce manual process errors by provide automated and a reliable attendance system uses face recognition technology.
- Increase privacy and security which student cannot presenting himself or his friend while they are not.

# TABLE OF CONTENTS

# TABLE OF FIGURES

# Chapter 1

# Introduction with important python's Library[1]

## 1.1 OpenCV

In this section, we will talk about OpenCV-Python (CV2) library, OpenCV functionality and some applications. OpenCV, the main library content should be interesting for the graduate students and researchers in image processing and computer vision areas, so Cv2 can do a lot of things, such as, it provides a lot of in-built primitives to handle operations related to image processing and Computer Vision, Video analysis, 3D reconstruction, Feature extraction, Object detection and Face and object recognition. If you want to learn more about this see reference.

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

## Applications of OpenCV

There Are lots of applications which are solved using OpenCV, some of them are listed below:

- Face Recognition.
- Automated Inspection and Surveillance.
- Number of People – count (Foot Traffic in A Mall, Etc.).
- Vehicle counting on highways along with their speeds.
- Interactive art installations.
- Anomaly (Defect) Detection in the manufacturing process (The Odd Defective Products).
- Street View Image Stitching.
- Video/Image search and retrieval.
- Robot and Driver-Less Car Navigation and Control.
- Object Recognition.
- Medical Image Analysis.
- Movies – 3D Structure from Motion.
- TV Channels advertisement recognition.

## OpenCV Functionality[2]

- Image/video I/O, processing, display (core, Image Processing, HighGui).
- Object/feature detection (OBJDETECT, features2d, NONFREE).
- Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab).
- Computational photography (photo, video, superres).
- Machine learning & clustering (ml, flann).
- CUDA acceleration (GBU).

## 1.2 Tkinter Programming [3]

In this section we will talk about Tk-tools python library and the type of its widgets, its imported as Tkinter module in the code, its used to create a GUI application as Creating a Calculator which would have a user-interface and functionalities that persists in a calculator and text-Editors, IDE's for coding are on a GUI app. so, its easily improve the appearance.

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

- Import the Tkinter module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

This would create a following window in Figure 1.1

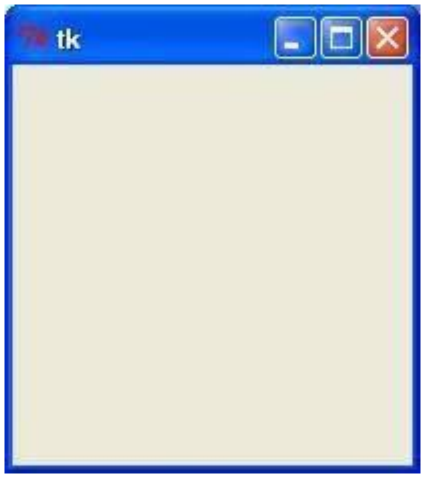


**Figure1.1 : main window**

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

## 1.3 CSV

Csv library is one of the most important libraries that enable us to deal with files such as excel, as we used it in most of the experiences to save names, appointments, etc. inside an Excel file

CSV (Comma Separated Values) is the most popular import and export format for spreadsheets and databases. The CSV format was used for many years before attempts to describe the format in a standardized way in RFC 4180. The lack of a well-defined standard means that often there are subtle differences in the data that different applications produce and consume. These differences can make processing CSV files from multiple sources annoying. However, while the delimiters and quote characters differ, the general format is similar enough that a single unit can be written that can efficiently process this data, hiding the details of reading and writing data from the programmer.

The Csv module implements classes for reading and writing tabular data in CSV format. It allows programmers to say, "write this data in the format that Excel prefers," or "read data from this file created by Excel," without knowing the exact details of the CSV format used by Excel. Programmers can also describe CSV formats that other applications understand or define their own CSV formats.

## 1.4 Numpy[4]

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
It also has functions for working in domain of linear algebra, Fourier transform, and matrices.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

## 1.5 Pillow

A lot of applications use digital images, and with this, there is usually a need to process the images used. If you are building your application with Python and need to add image processing features to it, there are various libraries you could use. Some popular ones are OpenCV, scikit-image, Python Imaging Library and Pillow.
We won't debate on which library is the best here, they all have their merits.
So, we will focus on Pillow library that is a power full, provides a wide array of image processing features, and it is simple to use.

The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool**.**

## 1.6 Pandas[5]

As we know, Python is a programming language provide a great platform for anyone starting in Machine Learning and AI to analyze and extract useful insights for businesses. Dealing with data for analysis and visualization is an imperative process in Machine Learning and Artificial Intelligence. So, Python has a library called “Pandas” - short for “Panel Data” (A panel is a 3D container of data) which contains in-built functions to clean, transform, manipulate, visualize and analyze data.
There are two major categories of data that you can come across while doing data analysis:

- One dimensional data
- Two-dimensional data

These data can be of any data type. Character, number or even an object.

Now, you will discover more information in a deep way about Pandas.

Pandas It is a Python package that provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet .
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a Pandas data structure.

The two primary data structures of pandas, Series (1-dimensional) and Data Frame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, Data Frame provides everything that R's data. Frame provides and much more. Pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating-point data.
- Size mutability: columns can be inserted and deleted from Data Frame and higher dimensional objects.
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, Data Frame, etc. automatically align the data for you in computations.

- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data.
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into Data Frame objects.
- Intelligent label-based slicing, fancy indexing, and sub setting of large data sets.
- Intuitive merging and joining data sets.
- Flexible reshaping and pivoting of data sets.
- Hierarchical labeling of axes (possible to have multiple labels per tick).
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format.
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

## 1.7 Time[6]

In our project “Face recognition-based attendance system”, it is important to know the time of the attending students. The teacher has to see who is late on the lecture. For this reason, we use the time module that is built-in Python. It is simple to use and provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code.
One of the ways you can manage the concept of Python time in the application is by using a floating-point number that represents the number of seconds that have passed since the beginning of an era—that is, since a certain starting point. Let’s dive deeper into what that means, why it’s useful, and how you can use it to implement logic, based on Python time, in the application.

Python has a module named time to handle time-related tasks. To use functions defined in the module, we need to import the module first. Here's how:

```
import time
```

**Python time. Time ()**

The time () function returns the number of seconds passed since epoch.
For Unix system, January 1, 1970, 00:00:00 at UTC is epoch (the point where time begins)

```
import time

seconds = time. Time () print ("Seconds since epoch =", seconds)
```

**Python time. ctime ()**

The time.ctime ()function takes seconds passed since epoch as an argument and returns a string representing local time.

```
import time
# seconds passed since epoch
seconds = 1545925769.9618232
local_time = time. ctime(seconds)print ("Local time:", local_time)
```

If you run the program, the output will be something like:

```
Local time: Thu Dec 27 15:49:29 2018
```

**Python time. Sleep ()**

The sleep() function suspends (delays) execution of the current thread for the given number of seconds.

```
import time
print ("This is printed immediately.")
time. Sleep (2.4) print ("This is printed after 2.4 seconds.")
```

# Chapter 2

# Classifiers

Detecting faces in images is a key step in numerous computer vision applications, such as face recognition or facial expression analysis. Automatic face detection is a difficult binary classification problem because of the large face intra-class variability which is due to the important influence of the environmental conditions on the face appearance. So, we need algorithm that sorts data into labeled classes, or categories of information. Classifiers are a concrete implementation of pattern recognition in many forms of machine learning. We will implement our use case using the Haar Cascade classifier.

## 2.1 HaarCascade

Face Detection using Haar Cascades Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images[8].

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

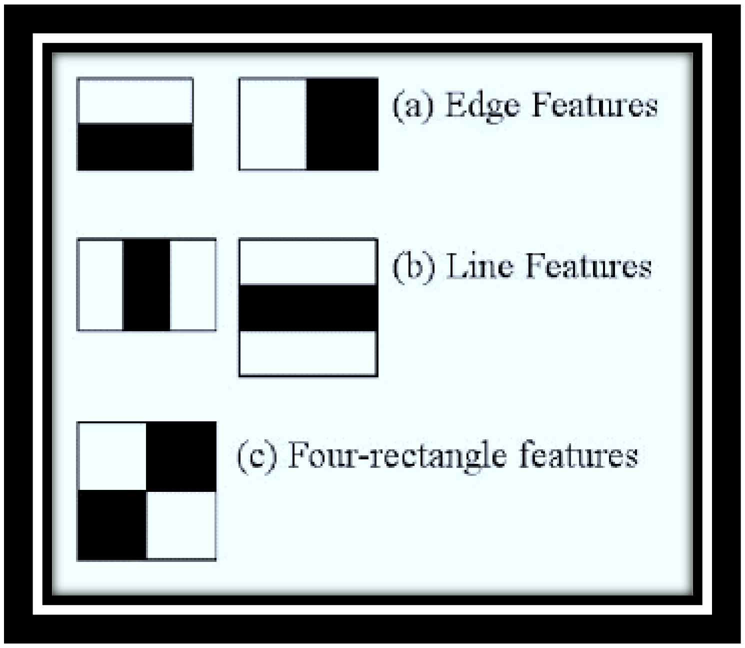


**Figure 2.1 : Haar features**

Now all possible sizes and locations of each kernel is used to calculate plenty of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by Adaboost**.**

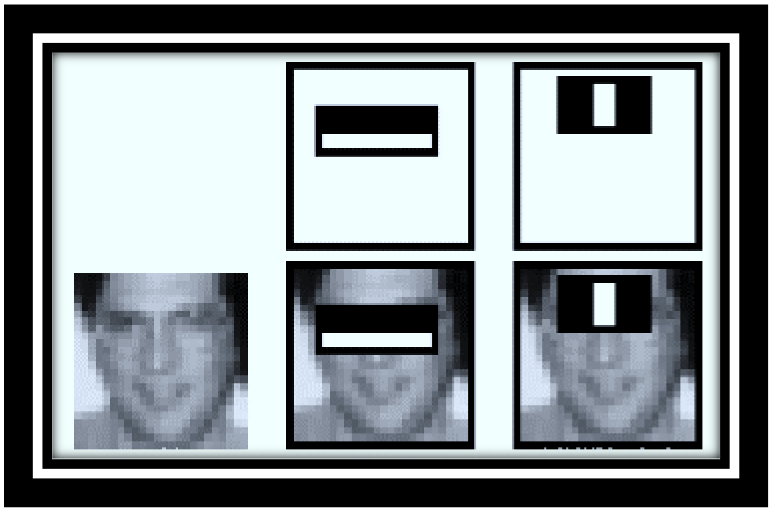


**Figure 2.2 : Face with Haar features**

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. Authors have a good solution for that.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don’t process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

For this they introduced the concept of Cascade of Classifiers**.** Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don’t consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

Authors’ detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in first five stages. (Two features in the above image is actually obtained as the best two features from Adaboost). According to authors, on an average, 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Read paper for more details or check out the references in Additional Resources section.

## 2.2 Haar-cascade Detection in OpenCV

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.

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 and eye detector with OpenCV.

First we need to load the required XML classifiers. Then load our input image (or video) in grayscale mode.

```
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Now we find the faces in the image. If faces are found, it returns the positions of detected faces as Rect(x,y,w,h). Once we get these locations, we can create a ROI for the face and apply eye detection on this ROI (since eyes are always on the face ).

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result looks like below :

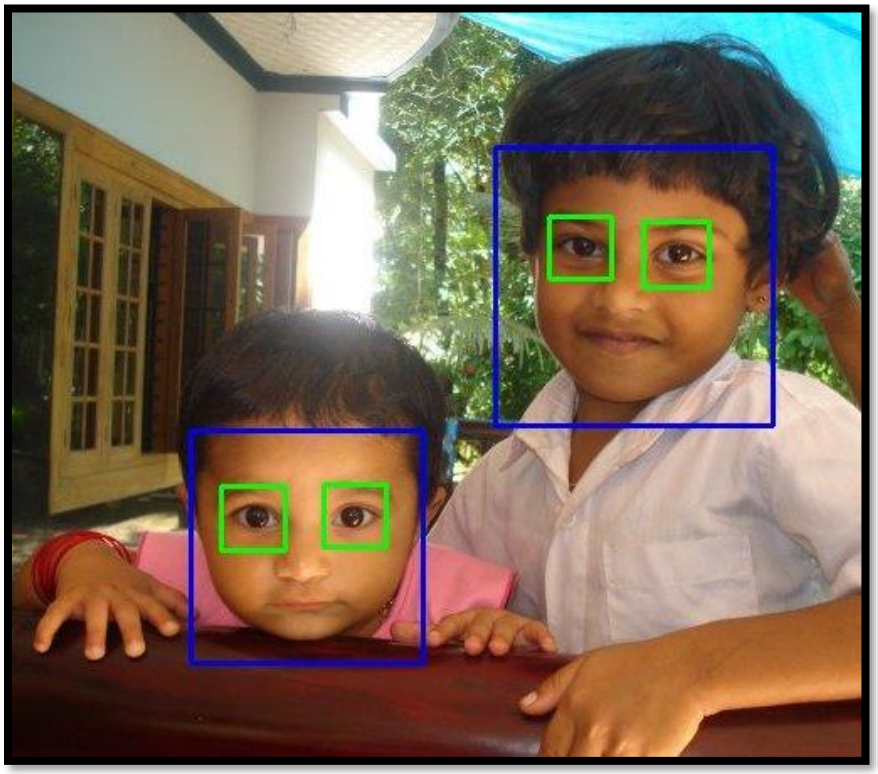


**Figure 2.3 : Face & eye detection**

## 2.3 LBPH Algorithm

After applying a face detection on a given frame(image), the face recognition algorithm tries to find some match of that extracted face with the ones in the database(trained samples/images) We will discuss LBPH algorithm .

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 [8].

**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:

**1.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 1.

- **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.

**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:

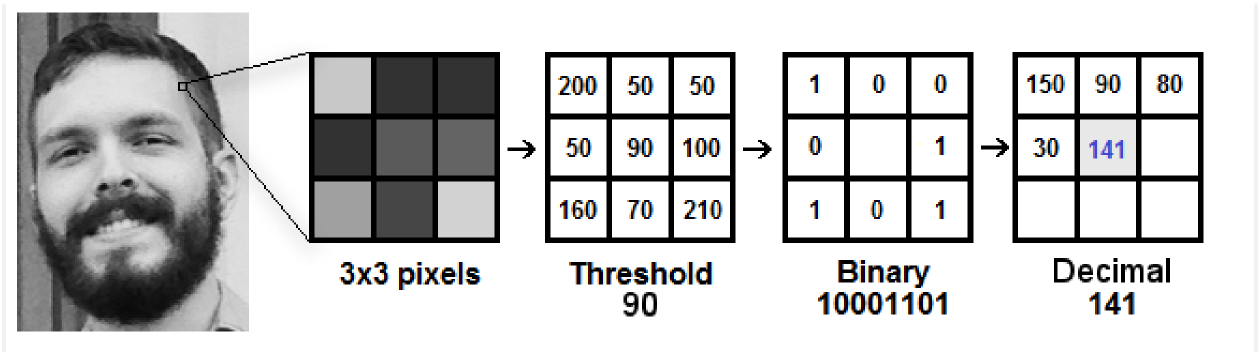


**Figure 2.4 : LBP calculations**

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.

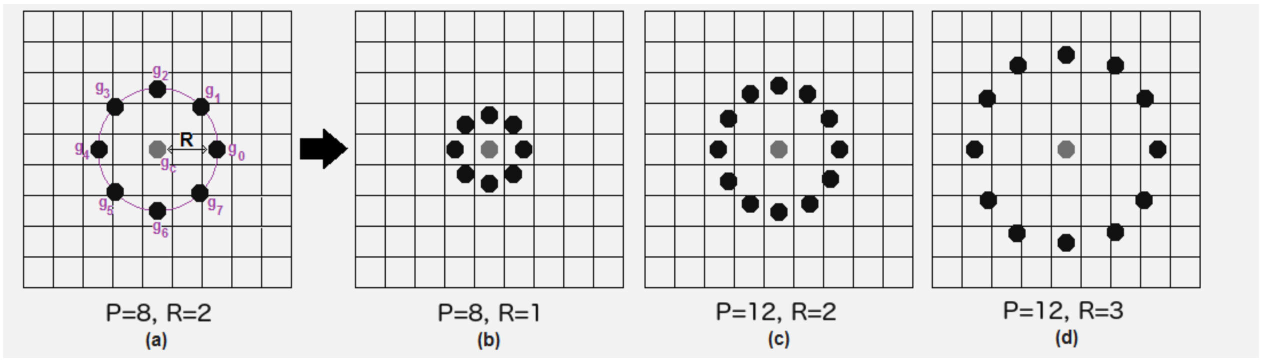


**Figure 2.5 : Circular LBP**

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:

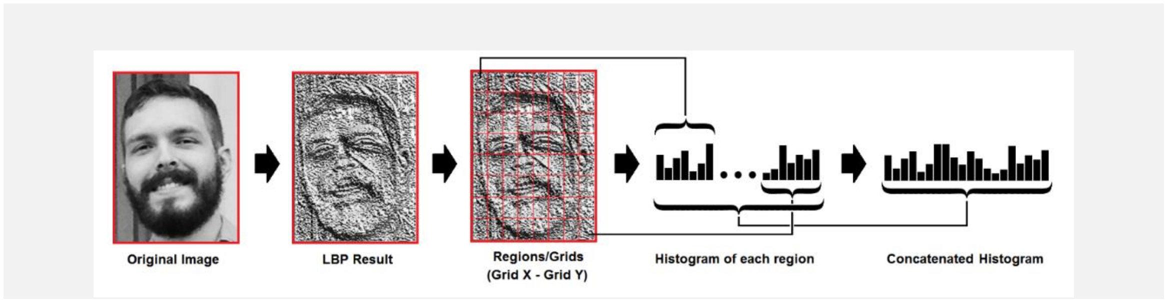


**Figure 2.6 : LBP Histograms**

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.

**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.

# Chapter 3

# Face Recognition experiments with code

## 3.1 face recognition - using picture[9]

in this experiment, we used the OpenCV library and the face recognition library, which are mainly aimed for recognizing faces and identifying people.
Where we do the following:

- The image of the person is placed inside the face folder
- We name the image with the person's name
- we name the image that we want the program to process with the test image
- The program compares it with the images inside the face folder
- If they are present, they are recognized and the name of the people is shown.

**Requirement:**

- cmake
- dlib
- face_recognition
- Numpy
- OpenCV-python

**Here is the code :**

1. we need to import all libraries we need

```
import library
import face_recognition as fr
import os
import cv2
import face_recognition
import numpy as np
from time import sleep
```

2. encode all faces

```
def get_encoded_faces():
    """
    looks through the faces folder and encodes all
    the faces

    :return: dict of (name, image encoded)
    """
    encoded = {}
    for dirpath, dnames, fnames in os.walk("./faces"):
        for f in fnames:
            if f.endswith(".jpg") or f.endswith(".png"):
                face = fr.load_image_file("faces/" + f)
                encoding = fr.face_encodings(face)[0]
                encoded[f.split(".")[0]] = encoding

    return encoded
```

3. encode face given the file name

```
def unknown_image_encoded(img):

    face = fr.load_image_file("faces/" + img)
    encoding = fr.face_encodings(face)[0]
    return encoding
```

4. we will find all of the faces in a given image and label them if it knows what they are , parameter im: str of file path , return: list of face names

```
def classify_face(im):
    faces = get_encoded_faces()
    faces_encoded = list(faces.values())
    known_face_names = list(faces.keys())
    img = cv2.imread(im, 1)
     face_locations = face_recognition.face_locations(img)
    unknown_face_encodings = face_recognition.face_encodings(img,     face_locations)

    face_names = []
    for face_encoding in unknown_face_encodings:
        # See if the face is a match for the known face(s)
        matches = face_recognition.compare_faces(faces_encoded, face_encoding)
        name = "Unknown"

        # use the known face with the smallest distance to the new face
        face_distances = face_recognition.face_distance(faces_encoded, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = known_face_names[best_match_index]

        face_names.append(name)

        for (top, right, bottom, left), name in zip(face_locations, face_names):
            # Draw a box around the face
            cv2.rectangle(img, (left-20, top-20), (right+20, bottom+20), (255, 0, 0), 2)

            # Draw a label with a name below the face
            cv2.rectangle(img, (left-20, bottom -15), (right+20, bottom+20), (255, 0, 0),
cv2.FILLED)
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(img, name, (left -20, bottom + 15), font, 1.0, (255, 255, 255), 2)

 # Display the resulting image
    while True:

        cv2.imshow('Video', img)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            return face_names
print(classify_face("test.jpg"))
```

Here are some of the results that the program processed and detected faces

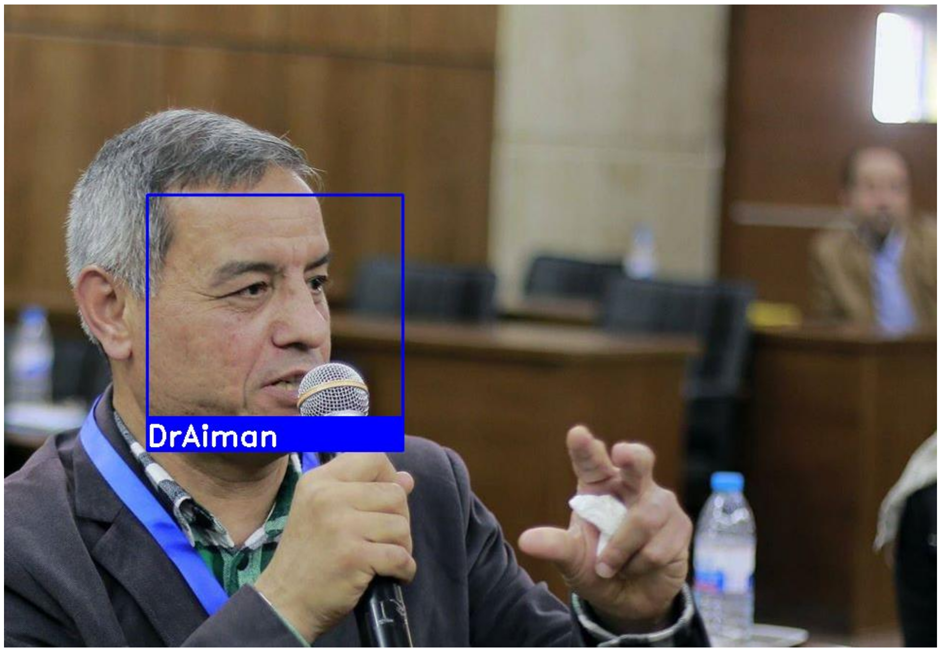


**Figure 3.1 : face detection results**

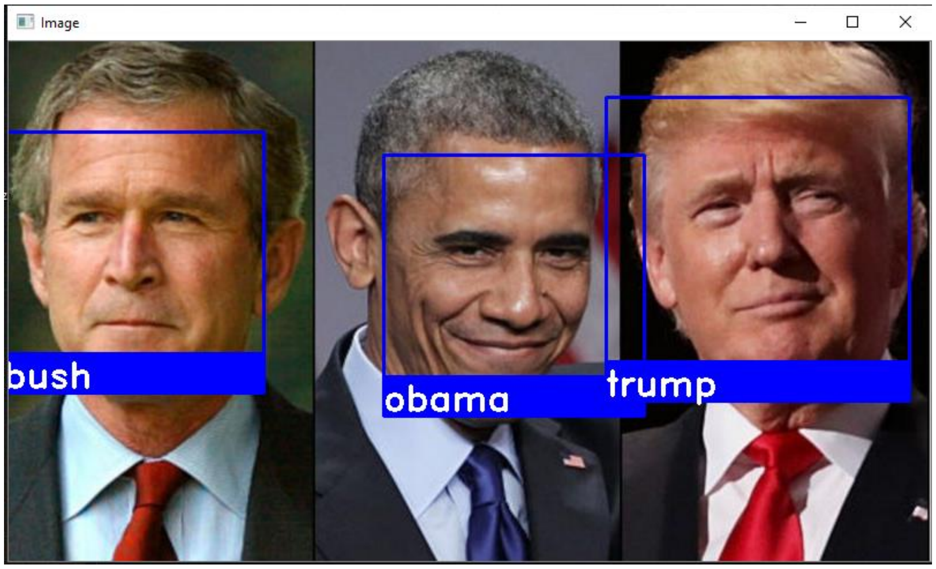


**Figure 3.2 : face detection results with multi faces**

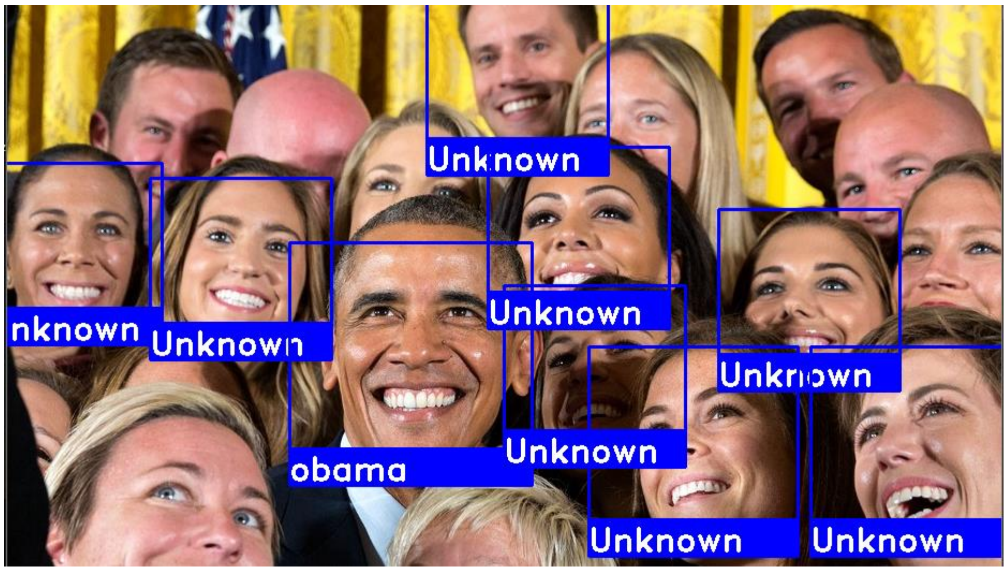


**Figure 3.3 : face detection results with unknown faces**

## 3.2 face recognition - using camera

The second experiment is exactly the same as the first one, except that the program processes the images captured from the camera[10]

1. we need to import all libraries we need

```
import face_recognition as fr
import os
import cv2
import face_recognition
import numpy as np
from time import sleep

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cap=cv2.VideoCapture(0)
_, img = cap.read()
```

2. looks through the faces folder and encodes all the faces

```
def get_encoded_faces():

    encoded = {}

    for dirpath, dnames, fnames in os.walk("./faces"):
        for f in fnames:
            if f.endswith(".jpg") or f.endswith(".png"):
                face = fr.load_image_file("faces/" + f)
                encoding = fr.face_encodings(face)[0]
                encoded[f.split(".")[0]] = encoding

    return encoded
```

3. encode face given the file name

```
def unknown_image_encoded(img):

    face = fr.load_image_file("faces/" + img)
    encoding = fr.face_encodings(face)[0]

    return encoding
```

4. will find all of the faces in a given image and label
   them if it knows what they are

```
def classify_face(im):

    faces = get_encoded_faces()
    faces_encoded = list(faces.values())
    known_face_names = list(faces.keys())

    face_locations = face_recognition.face_locations(img)
    unknown_face_encodings = face_recognition.face_encodings(img, face_locations)
```

4. will find all of the faces in a given image and label
   them if it knows what they are

```
face_names = []
    for face_encoding in unknown_face_encodings:
        # See if the face is a match for the known face(s)
        matches = face_recognition.compare_faces(faces_encoded, face_encoding)
        name = "Unknown"
        # use the known face with the smallest distance to the new face
        face_distances = face_recognition.face_distance(faces_encoded, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = known_face_names[best_match_index]
        face_names.append(name)

 for (top, right, bottom, left), name in zip(face_locations, face_names):
            # Draw a box around the face

            cv2.rectangle(img, (left-20, top-20), (right+20, bottom+20), (255, 0, 0), 2)

            # Draw a label with a name below the face

            cv2.rectangle(img, (left-20, bottom -15), (right+20, bottom+20), (255, 0, 0),
cv2.FILLED)
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(img, name, (left -20, bottom + 15), font, 1.0, (255, 255, 255), 2)

    # Display the resulting image
    while True:

        cv2.imshow('Video', img)
        if cv2.waitKey(25) & 0xFF == ord('q'):
         break

    cap.release()

    cv2.destroyAllWindows()

    return face_names

print(classify_face("test"))
```

Here are some of the results that the program processed and detected faces

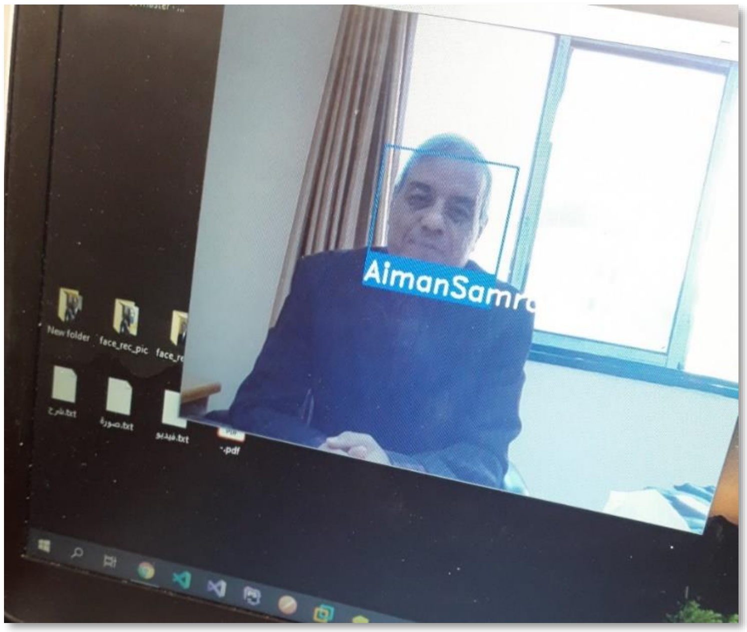


**Figure 3.4 : face detection results with an image captured from the camera**

## 3.3 Face detection with accuracy for every person in image

- Framework: Caffe deep learning

  Caffe deep learning framework it is have already two trains module for face detection.

- Libraries in code: cv2, numpy

- Module: DNN

**Summary of the code implementation:**

We have three required arguments:

1.image: The path to the input image.

2.prototxt: The path to the Caffe prototxt file.

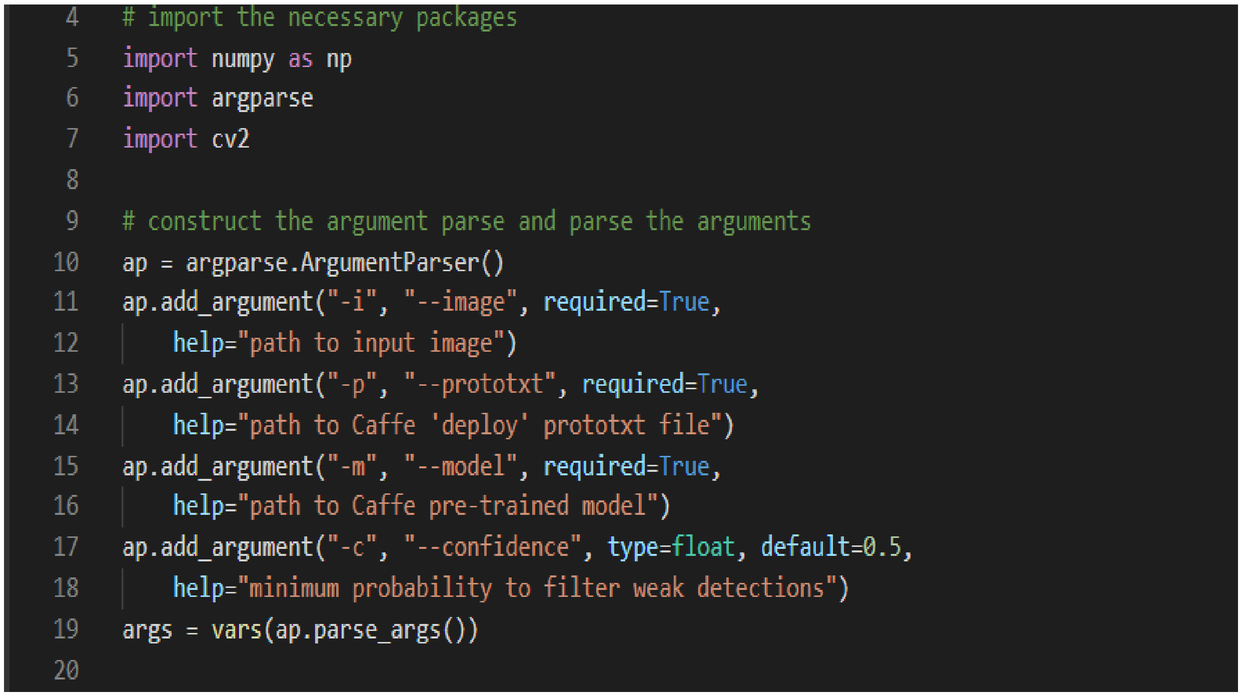
3.model: The path to the pretrained Caffe model.

An optional argument, --confidence, can overwrite the default threshold of 0.5.

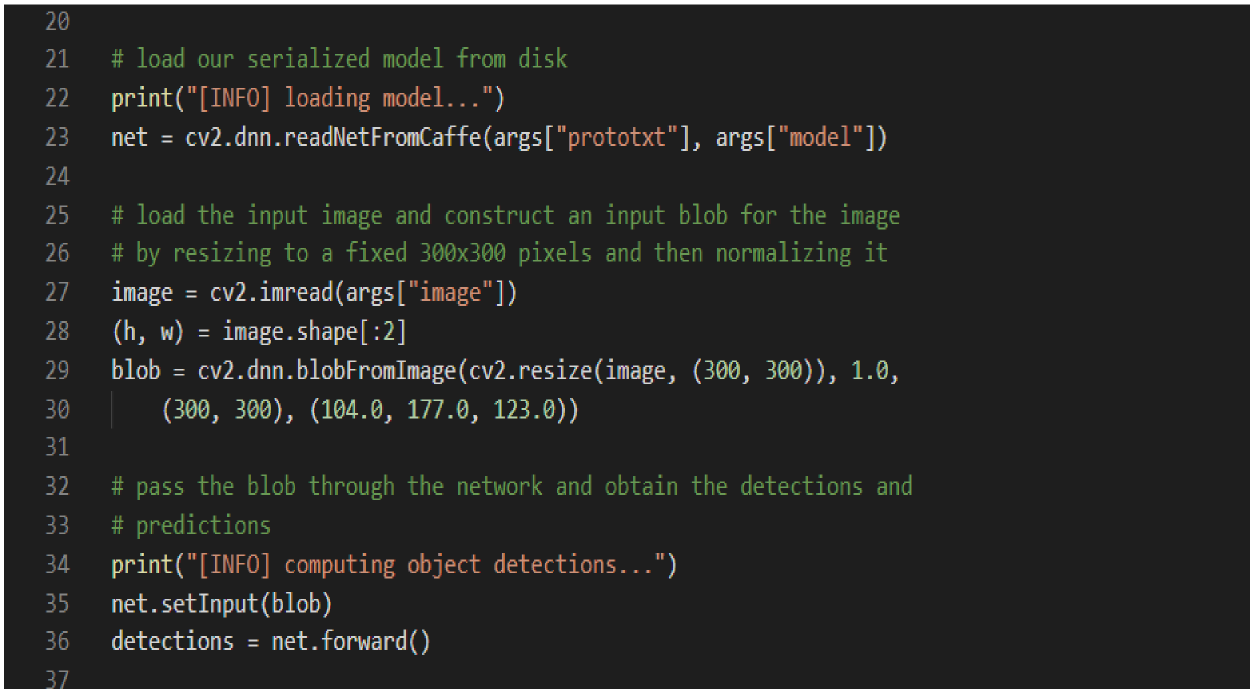
- First: we load our model using our --prototxt and --model file paths. We store the model as net.
- Then we load the image, extract the dimensions, and create a blob.
- The dnn. blobFromImage takes care of pre-processing which includes setting the blob dimensions and normalization.

Next, we'll apply face detection ,to detect faces, we pass the blob through the net,and from there we'll loop over the detections, and draw boxes around the detected faces. We can change the input by command in cmd.

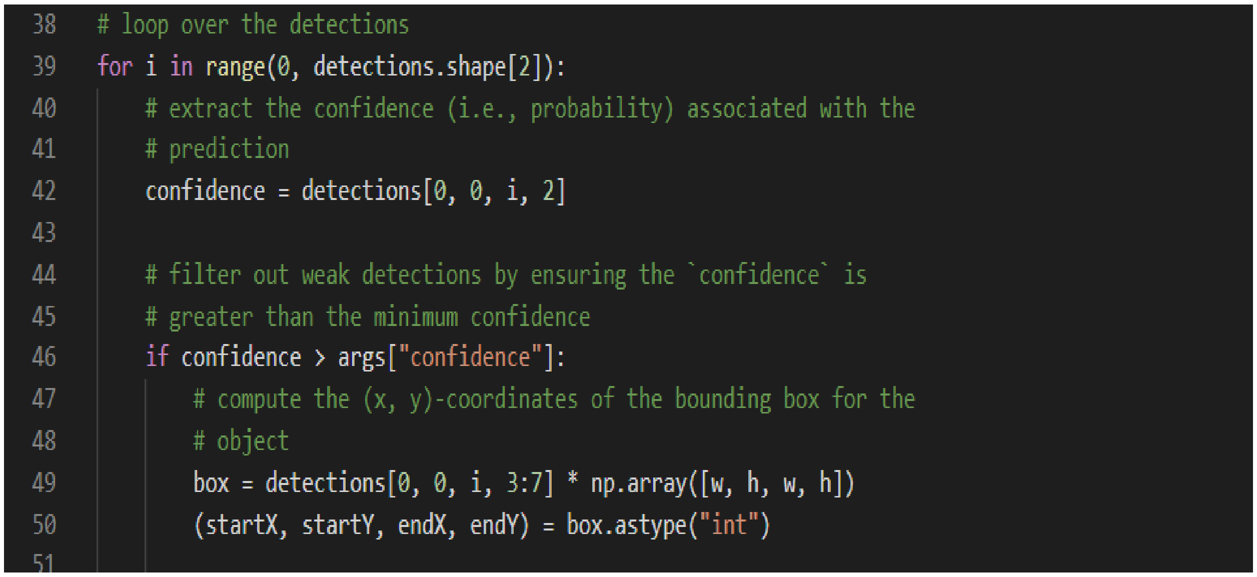
**Here is the code :**



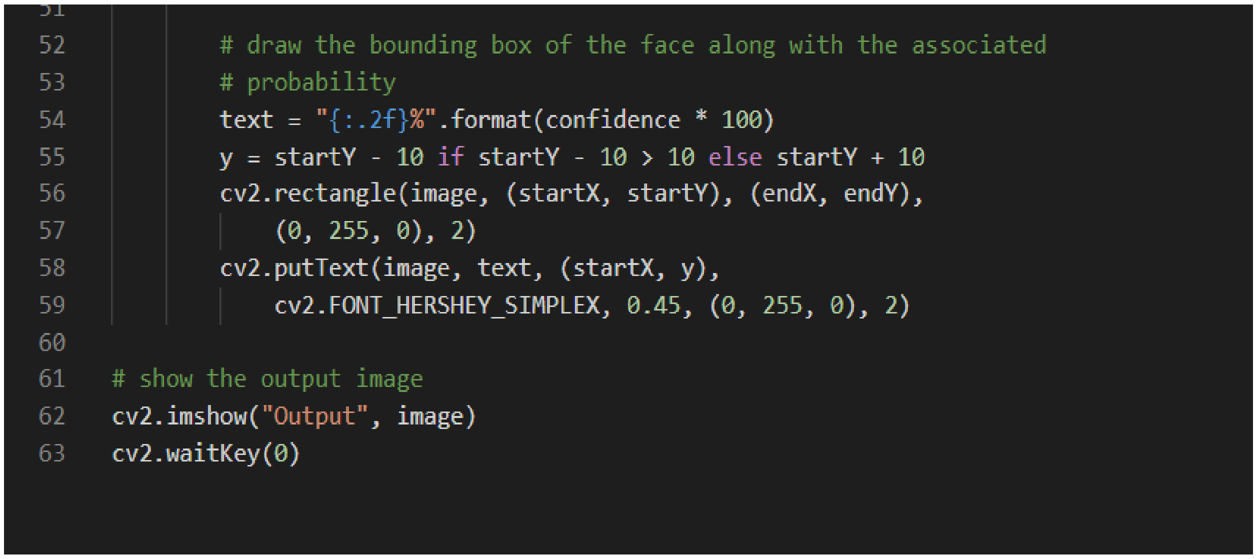
```
# import the necessary packages
import numpy as np
import argparse
import cv2

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-p", "--prototxt", required=True,
    help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
    help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

```



```

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# load the input image and construct an input blob for the image
# by resizing to a fixed 300x300 pixels and then normalizing it
image = cv2.imread(args["image"])
(h, w) = image.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,
    (300, 300), (104.0, 177.0, 123.0))

# pass the blob through the network and obtain the detections and
# predictions
print("[INFO] computing object detections...")
net.setInput(blob)
detections = net.forward()

```



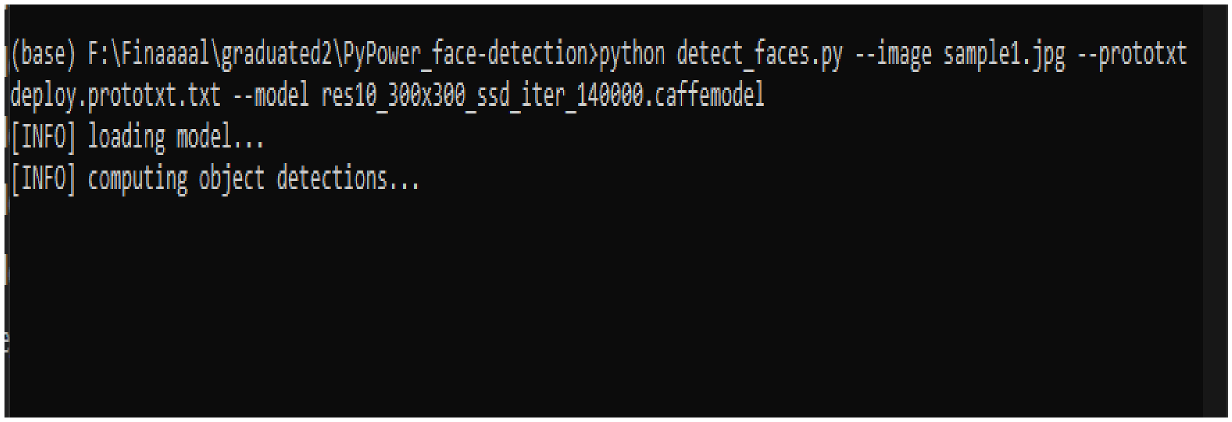
```python
# loop over the detections
for i in range(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated with the
    # prediction
    confidence = detections[0, 0, i, 2]

    # filter out weak detections by ensuring the `confidence` is
    # greater than the minimum confidence
    if confidence > args["confidence"]:
        # compute the (x, y)-coordinates of the bounding box for the
        # object
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

```



```python
        # draw the bounding box of the face along with the associated
        # probability
        text = "{:.2f}%".format(confidence * 100)
        y = startY - 10 if startY - 10 > 10 else startY + 10
        cv2.rectangle(image, (startX, startY), (endX, endY),
            (0, 255, 0), 2)
        cv2.putText(image, text, (startX, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)

# show the output image
cv2.imshow("Output", image)
cv2.waitKey(0)
```

## Code implementation:

We shall execute this command in cmd where sample1 is the name of image that contains faces:



```
(base) F:\Finaaaal\graduated2\PyPower_face-detection>python detect_faces.py --image sample1.jpg --prototxt
deploy.prototxt.txt --model res10_300x300_ssd_iter_140000.caffemodel
[INFO] loading model...
[INFO] computing object detections...
```

**Figure 3.5 : cmd execution code**

## Here are some results

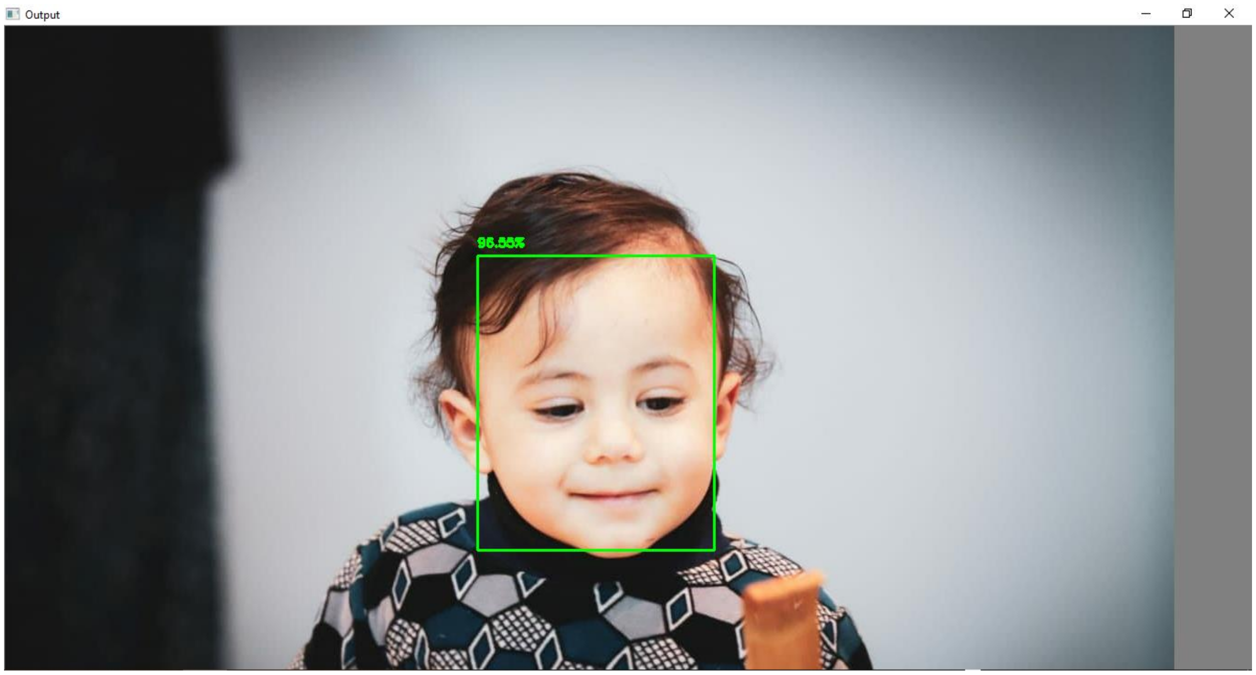


**Figure 3.6 : Face detection with accuracy**

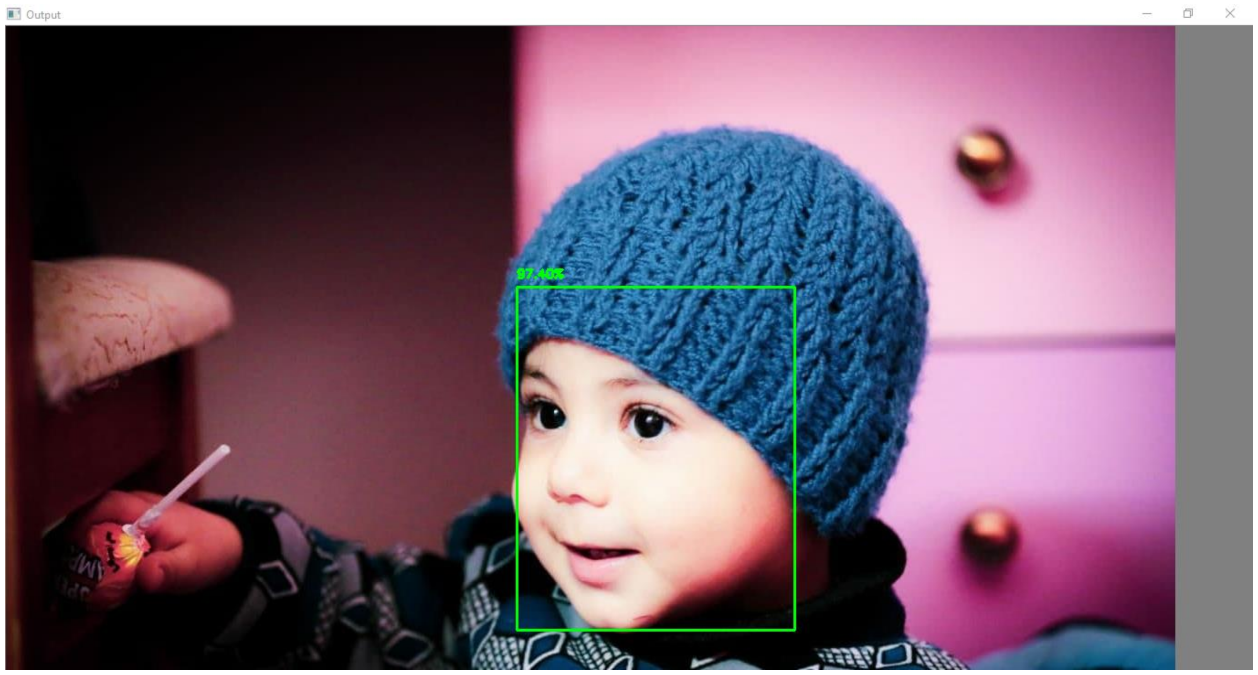


Figure 3.7: Face detection with accuracy

## 3.4 Face recognition by using hog model

The main idea of this experiment is recognizing the faces by comparing two folders. The first one is "known faces" that we can create many folders within it, such as, Kareem and Nada having their faces images in individual folders. The second folder is "unknown faces" that have images for detection.  These are faces that we intend to label (if any of the known faces exist in these images).
Hog model: "is based on extracting information about the edges in local regions of a target image and magnitude values of the pixels in an image. That is, it defines an image in terms of groups of local histograms that point to local regions of an image. The features of HOG can be seen on a grid of rose plots spaced uniformly. The grid dimensions depend upon the size of the cell and image. Thus, every rose plot depicts the gradient orientations distributed in a HOG cell."
That’s all the results we have reached on the experiment.

- It can recognize the faces from different angles.
- It can detect and recognize lots of pictures and have lots of faces when using Os to iterate the images inside the folders.
- We have got low accuracy in some images when using the social filters on the faces.

**Overview of the Face Recognition package:**

General Installation Information

1. Microsoft Visual Studio 2019
2. Download and install CMake
3. If we aimed for CUDA, let's now check if it compiled to use CUDA:
4. python >>> import dlib >>> dlib. DLIB_USE_CUDA True
5. Installing face_recognition: pip install
6. OpenCV -python
7. Model Hog

To begin, we'll need some samples of faces that we wish to detect and identify. We can do this task on a single image or with a video.

First, we'll create a directory called known faces, which will house directories of known identities. Inside of known faces, we will add a directory called folder for Obama, and folder for Trump Inside of here, we will have the following images:

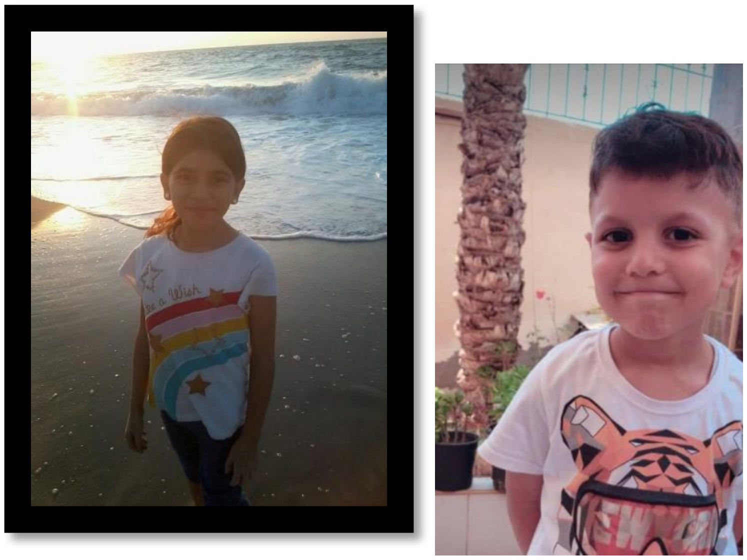


**Figure 3.8: images of known faces**

Second**,** we can then add a new directory called unknown faces. These are faces that we intend to label (if any of the known faces exist in these images.
We collect those pictures for different angles of their faces (Trump and Obama)

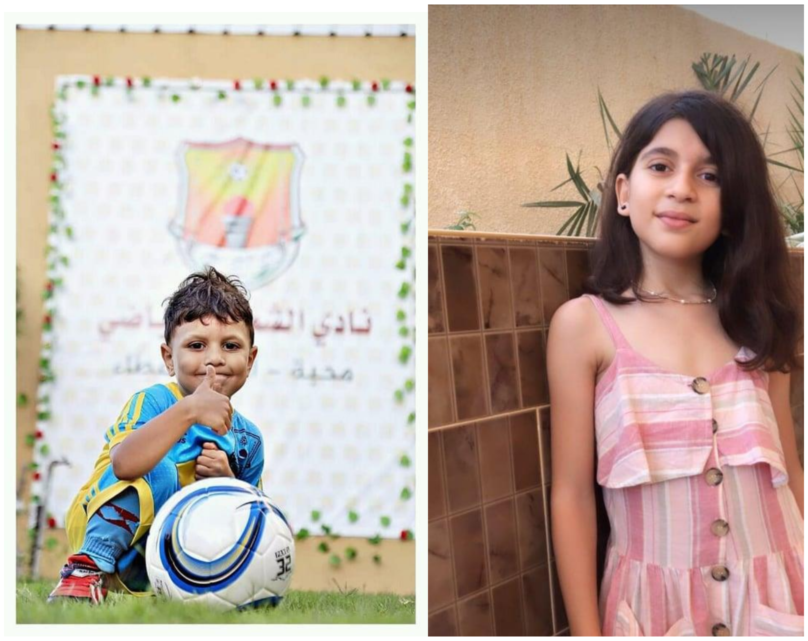


**Figure 3.9: images of Un_known faces**

**Now we can begin our code:**

```python
import face_recognition
import os
import cv2

KNOWN_FACES_DIR = 'Known_faces'
UNKNOWN_FACES_DIR = 'unknow_faces'
TOLERANCE = 0.6
FRAME_THICKNESS = 3
FONT_THICKNESS = 2
MODEL = 'hog'
```

Hog model: "is based on extracting information about the edges in local regions of a target image and magnitude values of the pixels in an image. That is, it defines an image in terms of groups of local histograms that point to local regions of an image.

The features of HOG can be seen on a grid of rose plots spaced uniformly. The grid dimensions depend upon the size of the cell and image. Thus, every rose plot depicts

We'll be using Os for working with directories and cv2 for labeling/drawing on our images.
The first two constants are just the names of our known and unknown directories.
Next, we have TOLERANCE. This is a value from 0 to 1 that will allow you to tweak the sensitivity of labeling/predictions. The default value here in the face_recognition package is 0.6. The lower the tolerance, the "stricter" the labels will be.
If you're getting a bunch of labels of some identity on a bunch of faces that aren't correct, you may want to lower the TOLERANCE. If you're not getting any labels at all, then you might want to raise the TOLERANCE.

The FRAME_THICKNESS value is how many pixels wide do you want the rectangles that encase a face to be and FONT_THICKNESS is how thick you want the font with the label to be.

Finally, you can choose what model to use. We'll use the cnn (convolutional neural network), but you can also use hog (histogram of oriented gradients) which is a non-deep learning approach to object detection.

The first things we need to do is prepare the faces that we intend to look for/identify. We'll start with a couple of lists. One for the faces, the other for the names associated.

```
print('Loading known faces...')
known_faces = []
known_names = []
```

Loading known faces

Now we iterate over our known faces directory, which contains possibly many directories of identities, which then contain one or more images with that person's face. From here, we want to load in this image with the face_recognition package, like:

```
for name in os.listdir(KNOWN_FACES_DIR):

    # Now we iterate over our known faces directory,
    # which contains possibly many directories of identities,
    # which then contain one or more images with that person's face.
    for filename in os.listdir(f'{KNOWN_FACES_DIR}/{name}'):
        # Load an image
        image =
face_recognition.load_image_file(f'{KNOWN_FACES_DIR}/{name}/{filename}
')
        # Get 128-dimension face encoding
        # Always returns a list of found faces, for this purpose we
take first face only
        # (assuming one face per image as you can't be twice on one
image)
        encoding = face_recognition.face_encodings(image)[0]

        # Append encodings and name
        known_faces.append(encoding)
        known_names.append(name)
```

At this point, we're ready to check unknown images for faces, and then to try to identify those faces. This loop will start in a familiar way:

```
print('Processing unknown faces...')
# Now let's loop over a folder of faces we want to label
for filename in os.listdir(UNKNOWN_FACES_DIR):

    # Load image
    print(f'Filename {filename}', end='')
    image = 
face_recognition.load_image_file(f'{UNKNOWN_FACES_DIR}/{filename}')
```

While known_images are just face shots, we assume that unknown images might have multiple people and other objects in them. Thus, we want to first locate those faces. We do that with:

```
locations = face_recognition.face_locations(image, model=MODEL)
```

Then we'd encode these images:

```
encodings = face_recognition.face_encodings(image, locations)
```

Notice that this method of encoding for the unknown faces is different than the encoding we used for the known face, which was: encoding = face_recognition.
face_encodings(image) [0]

It's expected that our known face is the only face in the image, so we're going with the first encoding, and we didn’t first grab locations because we don't really care about locations of faces in the known_images, but we want these locations in the unknown images so that we can label them.

Now we can iterate over the faces found in the unknown images, to see if we can find a match with any of our known faces. If we find one, we want to draw a rectangle around them. For that reason, we're going to use OpenCV to draw, and we'll first convert the image from RGB to BGR since OpenCV uses BGR. Doing that is as simple as:

```
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

This makes our current loop so far:

```
print('Processing unknown faces...')
# Now let's loop over a folder of faces we want to label
for filename in os.listdir(UNKNOWN_FACES_DIR):

    # Load image
    print(f'Filename {filename}', end='')
    image = 
face_recognition.load_image_file(f'{UNKNOWN_FACES_DIR}/{filename}')
    # While known_images are just face shots,
    #  we assume that unknown images might have multiple people and 
other objects in them. Thus,
    # we want to first locate those faces
    locations = face_recognition.face_locations(image, model=MODEL)
    encodings = face_recognition.face_encodings(image, locations)
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

Now we're going to iterate over each face found in the unknown image and check for any matches with our known faces:

```
print('Processing unknown faces...')
# Now let's loop over a folder of faces we want to label
for filename in os.listdir(UNKNOWN_FACES_DIR):
```

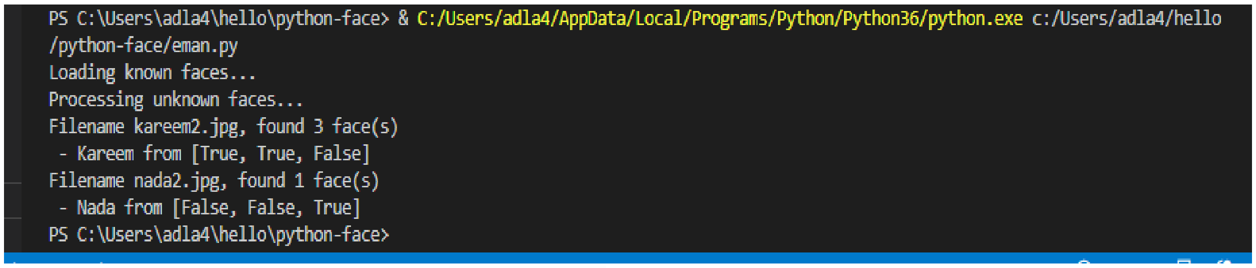
```
# Load image
print(f'Filename {filename}', end='')
image =
face_recognition.load_image_file(f'{UNKNOWN_FACES_DIR}/{filename}')
# While known_images are just face shots,
#  we assume that unknown images might have multiple people and other
objects in them. Thus,
# we want to first locate those faces
locations = face_recognition.face_locations(image, model=MODEL)
encodings = face_recognition.face_encodings(image, locations)
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
print(f', found {len(encodings)} face(s)')
for face_encoding, face_location in zip(encodings, locations):
    # We use compare_faces (but might use face_distance as well)
    # Returns array of True/False values in order of passed
known_faces
    results = face_recognition.compare_faces(known_faces,
face_encoding, TOLERANCE)
```

The results variable holds a list of Booleans, regarding if any matches were found:

```
print(results)
[False, False, False, True, False]
```

We take the index value for any of the Ture’s here, and look for the name from our known_names variable:

```
match = known_names [results. index (True)]print(match)
```



```
PS C:\Users\adla4\hello\python-face> & C:/Users/adla4/AppData/Local/Programs/Python/Python36/python.exe c:/Users/adla4/hello
/python-face/eman.py
Loading known faces...
Processing unknown faces...
Filename kareem2.jpg, found 3 face(s)
 - Kareem from [True, True, False]
Filename nada2.jpg, found 1 face(s)
 - Nada from [False, False, True]
PS C:\Users\adla4\hello\python-face>
```

**Figure 3.9: Matches faces**

Now, we want to draw a rectangle around this recognized face. To draw a rectangle in OpenCV, we need the top left and bottom right coordinates, and we use cv2.rectangle to draw it.

We also need a color for this box, and it would be neat to have this box color fairly unique to the identity.

Daniel (DhanOS) came up with the following code to take the first 3 letters in the string, and convert these to RGB values:

```
color = [(ord(c.lower()) - 97) * 8 for c in match[:3]]
print(color)
[144, 32, 104]
```

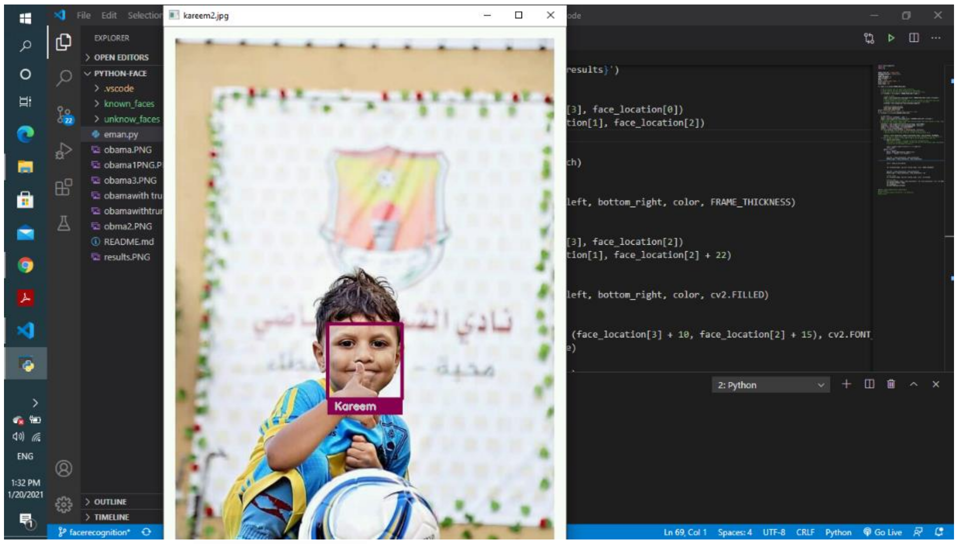
Which we'll convert to a function:

```
# Returns (R, G, B) from name
def name_to_color(name):
    # Take 3 first letters, tolower()
    # lowercased character ord() value rage is 97 to 122, substract
97, multiply by 8
    color = [(ord(c.lower()) - 97) * 8 for c in name[:3]]


return color
```

Beyond having a rectangle for the face itself, we'll add a smaller rectangle for the text for the identity, then of course place the text for that identity. The part just for that:

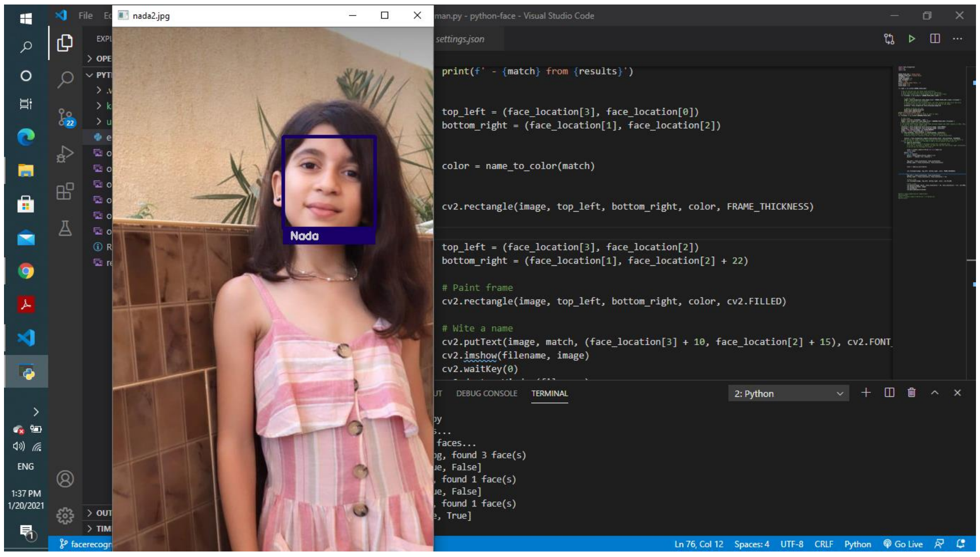
```

        if True in results:  # If at least one is true, get a name of
first of found labels
            match = known_names[results.index(True)]
            print(f' - {match} from {results}')

            # Each location contains positions in order: top, right,
bottom, left
            top_left = (face_location[3], face_location[0])
            bottom_right = (face_location[1], face_location[2])

            # Get color by name using our fancy function
            color = name_to_color(match)
            # Paint frame
            cv2.rectangle(image, top_left, bottom_right, color,
FRAME_THICKNESS)

            # Now we need smaller, filled grame below for a name
            # This time we use bottom in both corners - to start from
bottom and move 50 pixels down
            top_left = (face_location[3], face_location[2])
            bottom_right = (face_location[1], face_location[2] + 22)

            # Paint frame
            cv2.rectangle(image, top_left, bottom_right, color,
cv2.FILLED)

            # Wite a name
        cv2.putText(image, match, (face_location[3] + 10,
face_location[2] + 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    (200, 200, 200), FONT_THICKNESS)


    # Show image
    cv2.imshow(filename, image)
    cv2.waitKey(0)
cv2.destroyWindow(filename)
```

And here are results:



```
results}')

[3], face_location[0])
tion[1], face_location[2])

ch)

left, bottom_right, color, FRAME_THICKNESS)

[3], face_location[2])
tion[1], face_location[2] + 22)

left, bottom_right, color, cv2.FILLED)

 (face_location[3] + 10, face_location[2] + 15), cv2.FONT_
e)
```

Figure 3.10: Face recognition for Kareem


nada2.jpg
Nada
man.py - python-face - Visual Studio Code
settings.json
DEBUG CONSOLE
TERMINAL
2: Python
Ln 76, Col 12  Spaces: 4  UTF-8  CRLF  Python  Go Live

```
print(f' - {match} from {results}')


top_left = (face_location[3], face_location[0])
bottom_right = (face_location[1], face_location[2])


color = name_to_color(match)


cv2.rectangle(image, top_left, bottom_right, color, FRAME_THICKNESS)


top_left = (face_location[3], face_location[2])
bottom_right = (face_location[1], face_location[2] + 22)

# Paint frame
cv2.rectangle(image, top_left, bottom_right, color, cv2.FILLED)

# Wite a name
cv2.putText(image, match, (face_location[3] + 10, face_location[2] + 15), cv2.FONT_
cv2.imshow(filename, image)
cv2.waitKey(0)
```

```
py
...
 faces...
g, found 3 face(s)
e, False]
, found 1 face(s)
e, False]
, found 1 face(s)
, True]
```

Figure 3.11: *Face recognition for Nada*

# Chapter 4

# Face Recognition based attendance system

**Purpose**

The basic purpose of this experiment is to take attendance using face detection. This program makes CSV file of present attendees automatically After successful face detection. Also, it will make a CSV file for registered student's info.

**libraries:**

This program using these libraries. OpenCV, Numpy, OS, Pandas, Tkinter.

**Features:**

1) Easy to use with interactive GUI support.
2) Password protection for new person registration.
3) Creates/Updates CSV file for details of students on registration.
4) Creates a new CSV file every day for attendance and marks attendance with proper date and time.
5) Displays live attendance updates for the day on the main screen in tabular format with Id, name, date and time.

## Flow diagram

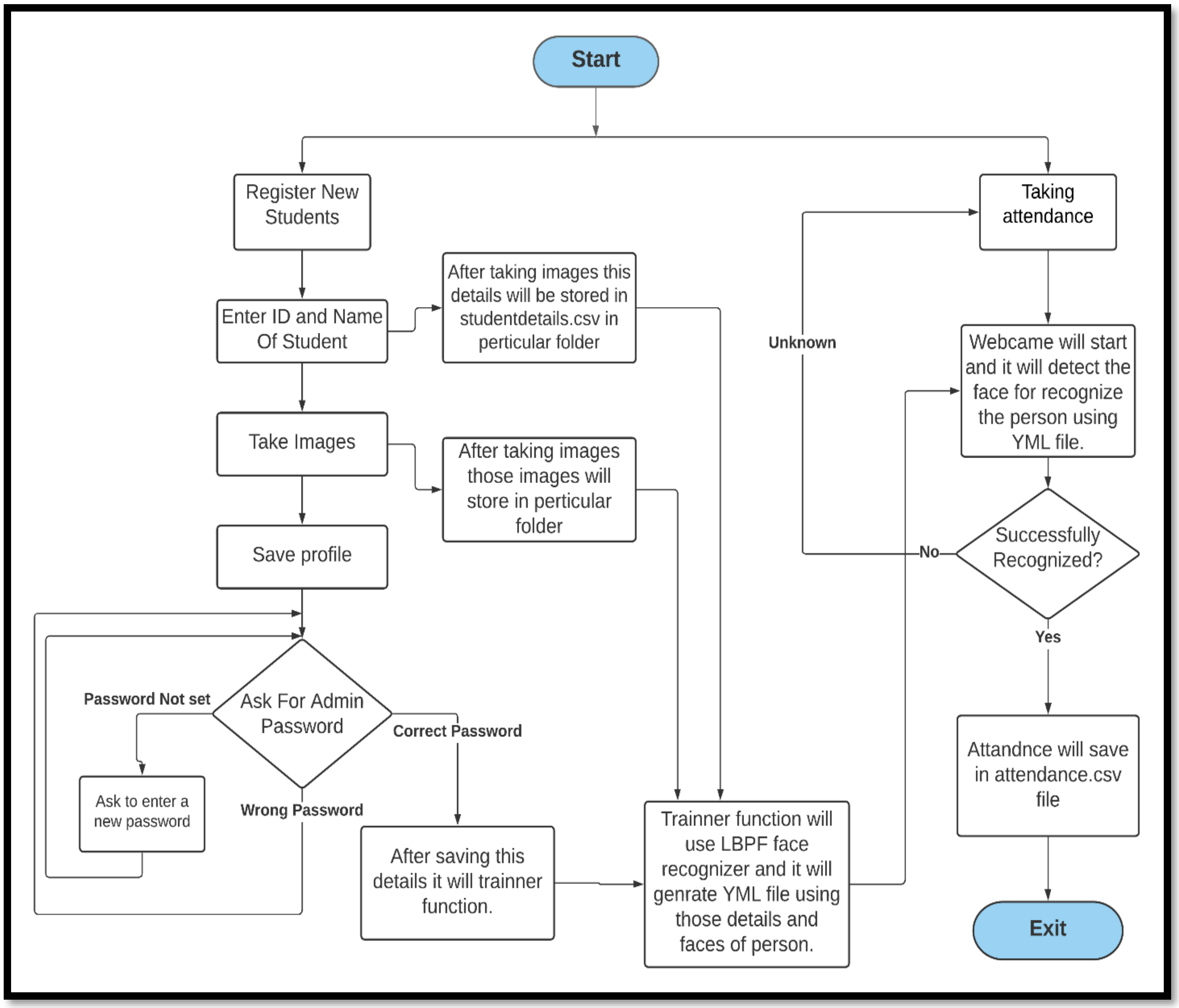

Start
Register New Students
Taking attendance
Enter ID and Name Of Student
After taking images this details will be stored in studentdetails.csv in perticular folder
Unknown
Webcame will start and it will detect the face for recognize the person using YML file.
Take Images
After taking images those images will store in perticular folder
Save profile
No
Successfully Recognized?
Yes
Password Not set
Ask For Admin Password
Correct Password
Ask to enter a new password
Wrong Password
After saving this details it will trainner function.
Trainner function will use LBPF face recognizer and it will genrate YML file using those details and faces of person.
Attandnce will save in attendance.csv file
Exit

**Figure 4.1: Flow diagram of attendance system**

**Part 1 : GUI**

In this project, we made one simple GUI using the python Tkinter library so that the user can easily use this project without any backend knowledge. Tkinter is the standard GUI library for Python.

For making this GUI we mainly used Tkinter's frame, menubar, button, label, message box, table, textbox, etc. need to divide main screen into two parts which are nothing but frames. one is for Registration and the second is for taking attendance.

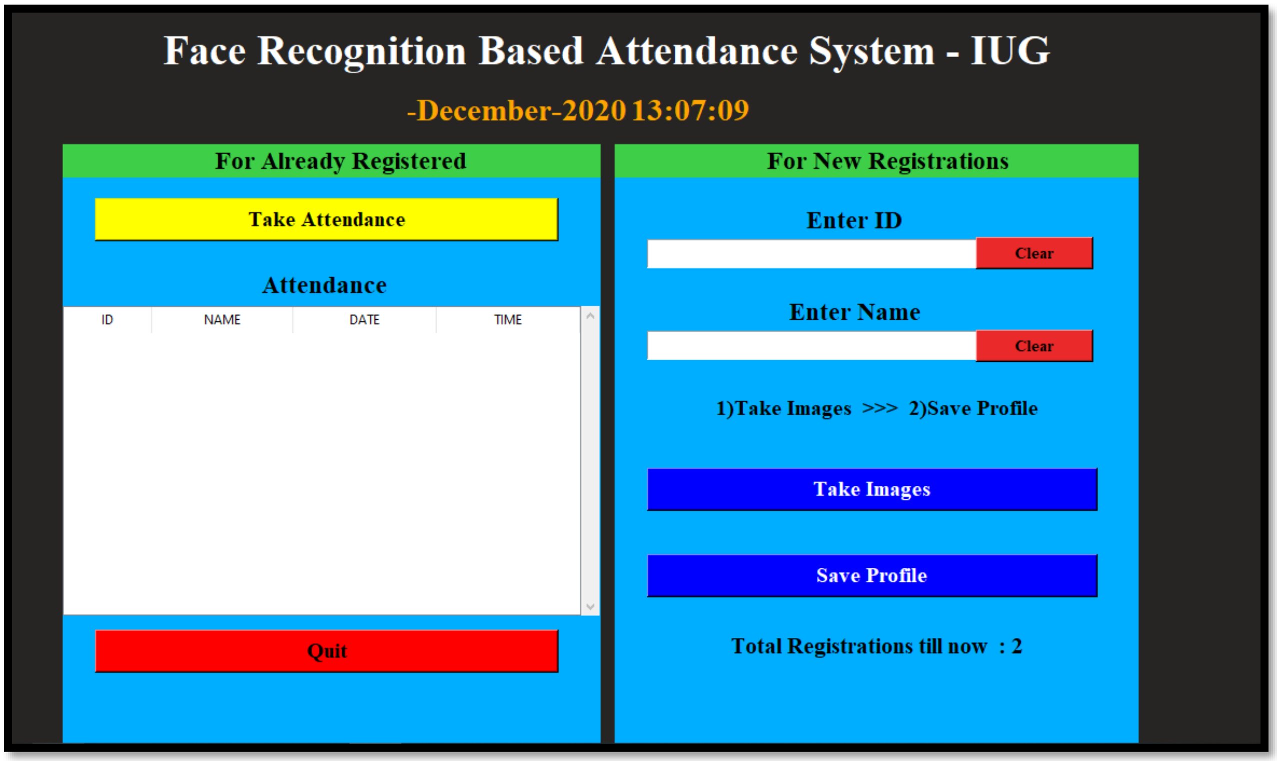


**Figure 4.2: GUI of the project attendance system**

As shown in the above image this GUI will help users to register new students as well as to take attendance. At the bottom, it will also show you the total number of registered students.

This GUI window also contains menubar with two sub-menu Help and About. The help menu contains 3 commands contact me, change Admin Password, exit.

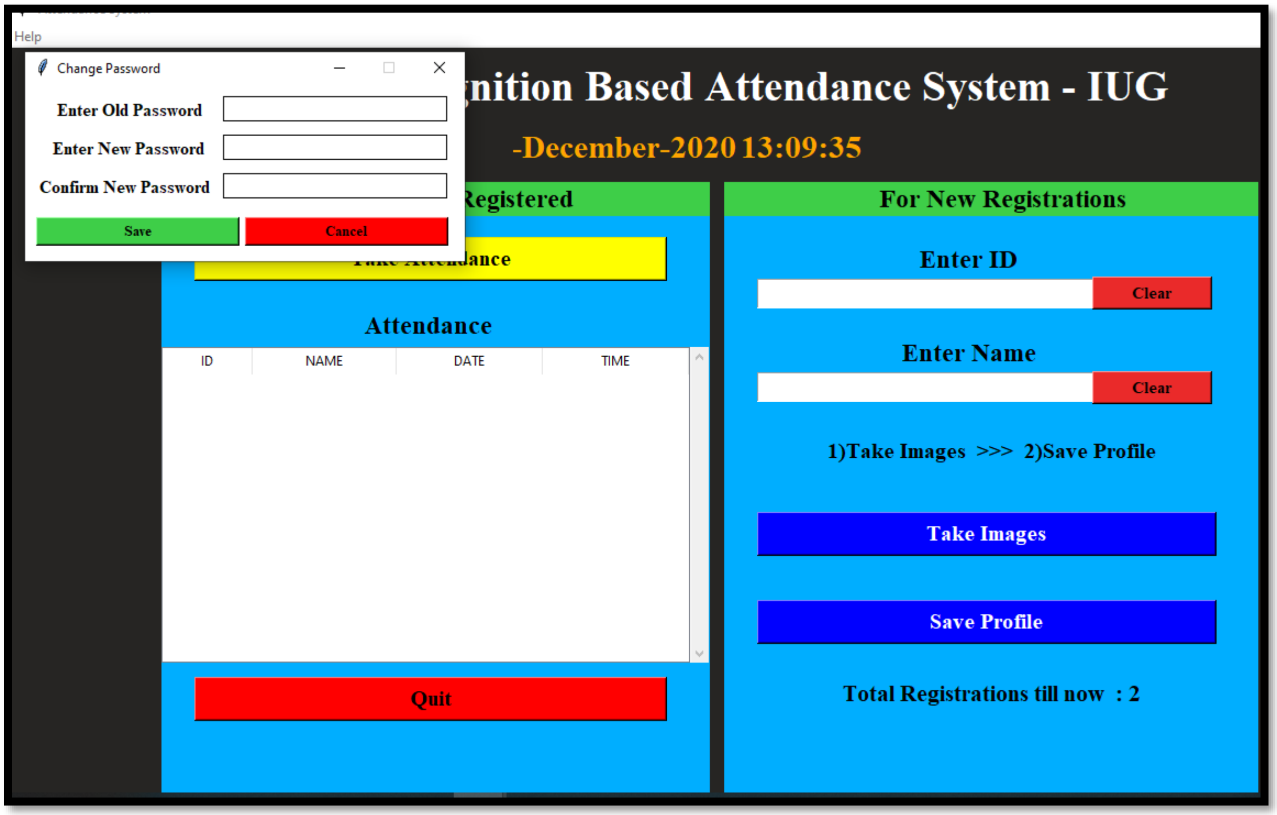


**Figure 4.3: Menu bar - Help of GUI**

## Part 2 : Take Images

when users register for new students it will take 100 images of that student and save these images to one folder which will be created at the time of first registration. we use OpenCV to take images and detection. OpenCV is a library of programming functions mainly aimed at real-time computer vision. we used a cascade classifier of OpenCV for face detection. To use this cascade classifier, we need the haarcascade_frontalface_default.xml file which includes all the Haar cascade features of a face.

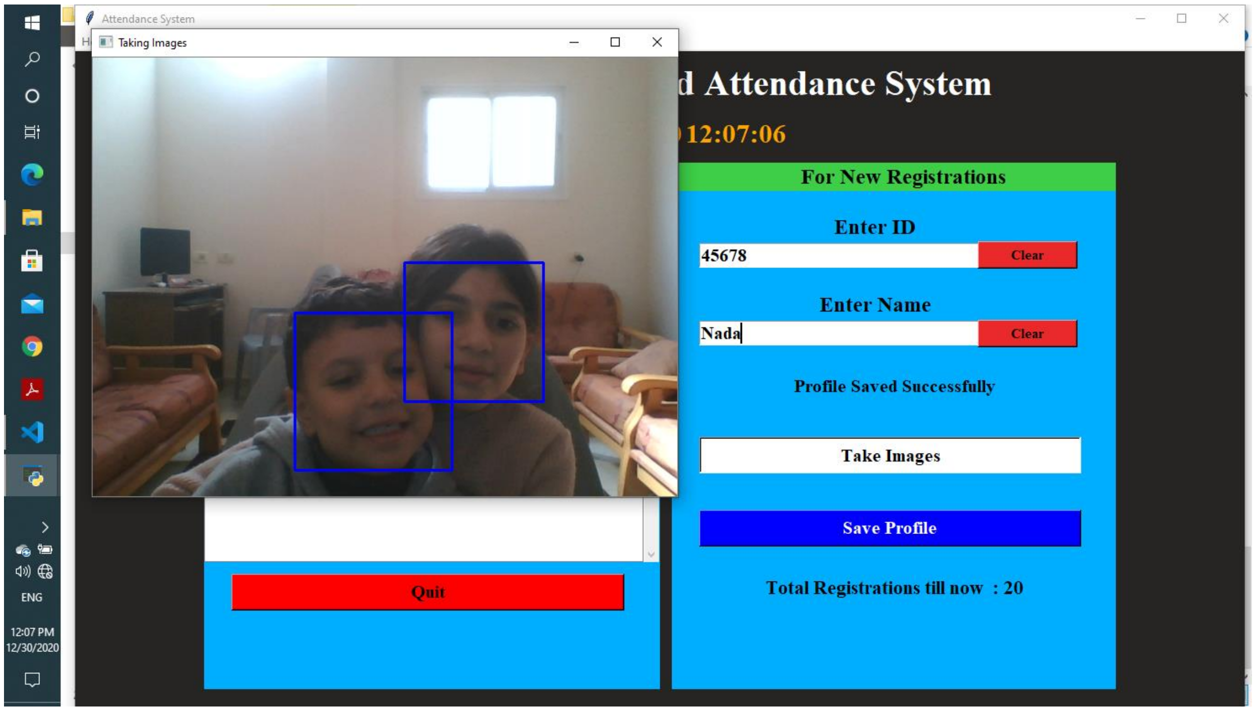


**Figure 4.4: Taking images - Face detection**

When users want to register new students, they have to enter first details like ID and Name. After that, they have to take images of the student for that when the user presses the button ‘Take’.

Images’ webcam will start and it will take 100 images of the student. There are some methods in OpenCV to take Images. using videoframe of OpenCV we're taking frames and from that, we're extracting only face images using cascade classifier. After taking images we will store those images in one folder.

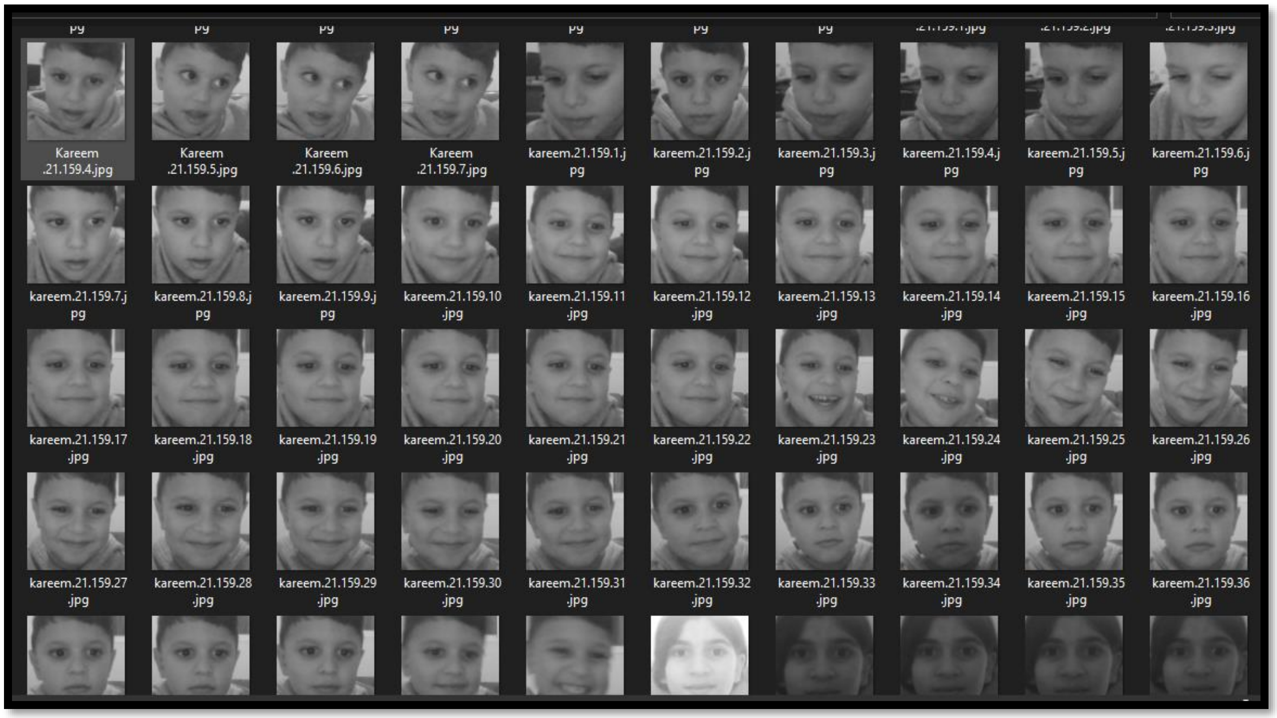


**Figure 4.5: Folder for training images**

**Part 3 : Save profile**

When images will be taken for any student user needs to click the ‘save profile’ button so that it will ask for the admin password. If the admin password was not set it will ask you first to set the admin password or else you need to enter the admin password. This password will be saved in one plan txt file.

If this password is correct then it will call the trainer function which will be going to generate a YML file and it will train our LBPH recognizer using those 100 images. This YML file and password txt file will save in one different folder name “pass_train”.

After saving the profile there are 3 different folders generated in the current directory. Out of that 3 folders, one is containing images of students, the second one is containing a CSV file of student’s details and the third one is containing a pass.txt file and YMLfile.

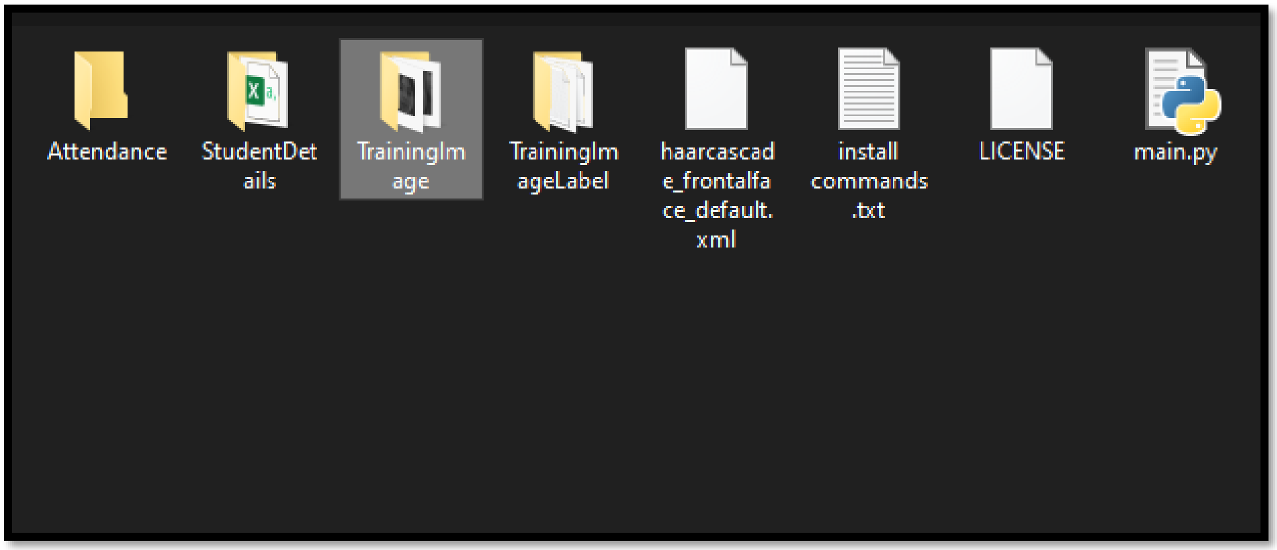


**Figure 4.6 : Folders of images’s students and csv file**

**Part 4 : Taking attendence**

When a user wants to take attendance and press the ‘take attendance’ button webcam will start and one videoframe window will generate to recognize the faces using the YML file. If the face will successfully recognize then it will put the name of the person at the bottom of the rectangle which is showing the detected face area.

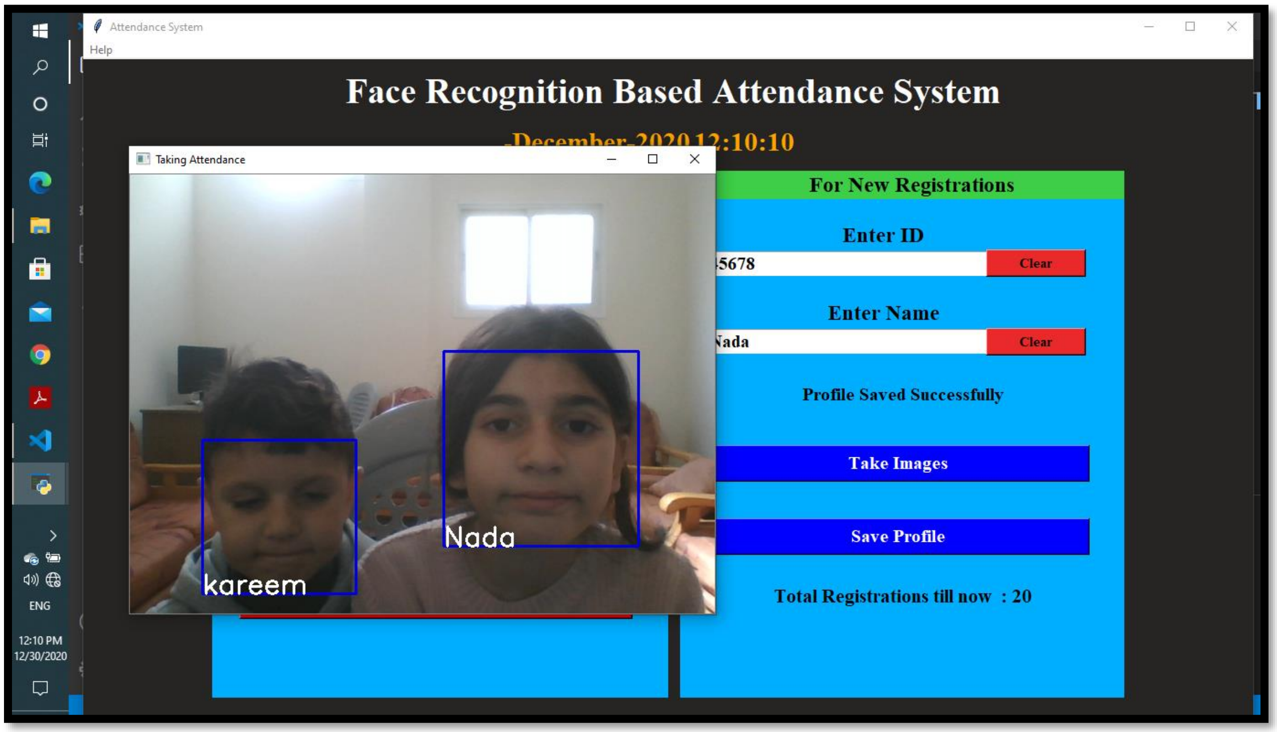


**Figure 4.7 : Face recognition - Taking attendance for two students**

After successfully recognized the attendance will be shown in the table in the 2nd frame and the Attandance.csv file will be generated in a particular folder. This CSV file contains the student's ID, name along with time at which attendance was taken for that student.

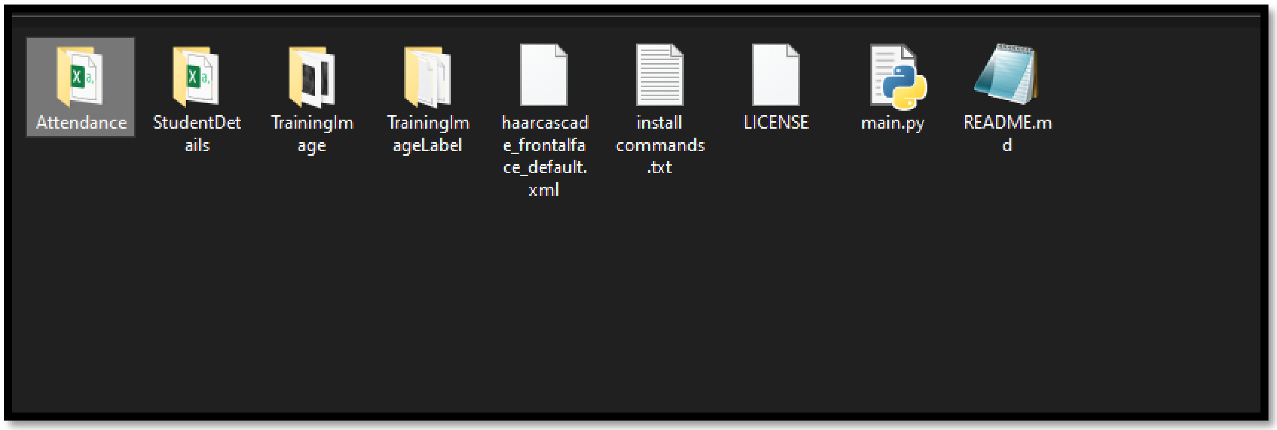
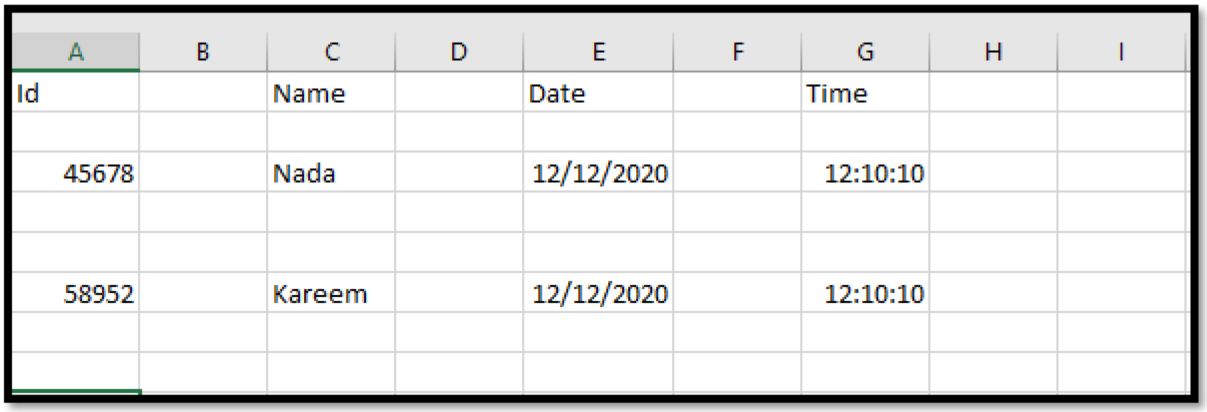


**Figure 4.8 :path of csv file for attendance student**

Attendance will store in one CSV file. This CSV file will be created datewise i.e attendance of one day is store in one CSV file. If a person will not recognize then in a video capturing it will show unknown. Let's look at the CSV file of attendance.



| A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| Id | | Name | | Date | | Time | | |
| | | | | | | | | |
| 45678 | | Nada | | 12/12/2020 | | 12:10:10 | | |
| | | | | | | | | |
| | | | | | | | | |
| 58952 | | Kareem | | 12/12/2020 | | 12:10:10 | | |
| | | | | | | | | |
| | | | | | | | | |

**Figure 4.9: csv file for attendance student**

## Validation

The project trains 100 images for one person from different angles, and it recognizes the facial features.

Cascade classifier could recognize the features of the face from the Clear lighting but sometimes it couldn’t recognize the features of the face from the dark theme. In addition, the distance between the face and camera must be short for recognition. When we tried to take a picture from far away from the camera, the detection was defined as the face as unknown.  The project could detect many faces with an accuracy of roughly 80%.

## Code

1) We will import OpenCV library for image processing, opening the webcam etc

- Os is required for managing files like directories
- Numpy is basically used for matrix operations
- PIL is Python Image Library

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mess
import tkinter.simpledialog as tsd
import cv2,os
import csv
import numpy as np
from PIL import Image
import pandas as pd
import datetime
import time
```

2) Method for checking existence of path i.e the directory.

```
def assure_path_exists(path):
    dir = os.path.dirname(path)
    if not os.path.exists(dir):
        os.makedirs(dir)
```

3) Method for getting the current local time from the PC.

```
def tick():

    time_string = time.strftime('%H:%M:%S')

    clock.config(text=time_string)

    clock.after(200,tick)
```

4) Method for displaying Contact menu.

```
def contact():
    mess._show(title='Contact us', message="Please contact us on : 
'student@gmail.com' ")
```

5) Method for checking if HaarCascade file exists.

```
def check_haarcascadefile():
    exists = os.path.isfile("haarcascade_frontalface_default.xml")
    if exists:
        pass
    else:
        mess._show(title='Some file missing', message='Please contact us for help')
        window.destroy()
```

6) Method for saving password.

```
def save_pass():
    assure_path_exists("TrainingImageLabel/")
    exists1 = os.path.isfile("TrainingImageLabel\psd.txt")
    if exists1:
        tf = open("TrainingImageLabel\psd.txt", "r")
        key = tf.read()
    else:
        master.destroy()
        new_pas = tsd.askstring('Old Password not found', 'Please enter a new password
below', show='*')
        if new_pas == None:
            mess._show(title='No Password Entered', message='Password not set!! Please
try again')
        else:
            tf = open("TrainingImageLabel\psd.txt", "w")
            tf.write(new_pas)
            mess._show(title='Password Registered', message='New password was
registered successfully!!')
            return
    op = (old.get())
    newp= (new.get())
    nnewp = (nnew.get())
    if (op == key):
        if(newp == nnewp):
            txf = open("TrainingImageLabel\psd.txt", "w")
            txf.write(newp)
        else:
            mess._show(title='Error', message='Confirm new password again!!!')
            return
    else:
        mess._show(title='Wrong Password', message='Please enter correct old
password.')
        return
    mess._show(title='Password Changed', message='Password changed successfully!!')
    master.destroy()
```

7) Method for changing password.

```
def change_pass():
    global master
    master = tk.Tk()
    master.geometry("400x160")
    master.resizable(False,False)
    master.title("Change Password")
    master.configure(background="white")
    lbl4 = tk.Label(master,text='    Enter Old Password',bg='white',font=('times', 12, ' bold
'))
    lbl4.place(x=10,y=10)
    global old
    old=tk.Entry(master,width=25 ,fg="black",relief='solid',font=('times', 12, ' bold
'),show='*')
    old.place(x=180,y=10)
    lbl5 = tk.Label(master, text='   Enter New Password', bg='white', font=('times', 12, '
bold '))
    lbl5.place(x=10, y=45)
    global new
    new = tk.Entry(master, width=25, fg="black",relief='solid', font=('times', 12, ' bold
'),show='*')
    new.place(x=180, y=45)
    lbl6 = tk.Label(master, text='Confirm New Password', bg='white', font=('times', 12, '
bold '))
    lbl6.place(x=10, y=80)
    global nnew
    nnew = tk.Entry(master, width=25, fg="black", relief='solid',font=('times', 12, ' bold
'),show='*')
    nnew.place(x=180, y=80)
    cancel=tk.Button(master,text="Cancel", command=master.destroy ,fg="black"
,bg="red" ,height=1,width=25 , activebackground = "white" ,font=('times', 10, ' bold '))
    cancel.place(x=200, y=120)
    save1 = tk.Button(master, text="Save", command=save_pass, fg="black",
bg="#3ece48", height = 1,width=25, activebackground="white", font=('times', 10, ' bold
'))
    save1.place(x=10, y=120)
    master.mainloop()
```

8) method for displaying password window.

```
def psw():
    assure_path_exists("TrainingImageLabel/")
    exists1 = os.path.isfile("TrainingImageLabel\psd.txt")
    if exists1:
        tf = open("TrainingImageLabel\psd.txt", "r")
        key = tf.read()
    else:
        new_pas = tsd.askstring('Old Password not found', 'Please enter a new password
below', show='*')
        if new_pas == None:
            mess._show(title='No Password Entered', message='Password not set!! Please
try again')
        else:
            tf = open("TrainingImageLabel\psd.txt", "w")
            tf.write(new_pas)
            mess._show(title='Password Registered', message='New password was
registered successfully!!')
            return
    password = tsd.askstring('Password', 'Enter Password', show='*')
    if (password == key):
        TrainImages()
    elif (password == None):
        pass
    else:
        mess._show(title='Wrong Password', message='You have entered wrong
password')
```

9) Method for cleaning the first text input.

```
def clear():
    txt.delete(0, 'end')
    res = "1)Take Images  >>>  2)Save Profile"
    message1.configure(text=res)
```

10) Method for cleaning the second text input.

```
def clear2():
    txt2.delete(0, 'end')
    res = "1)Take Images  >>>  2)Save Profile"
    message1.configure(text=res)
```

11) Method for displaying taking image from video capture and then saving the captured face in the dataset folder Training Image.

```
def TakeImages():
    check_haarcascadefile()
    columns = ['SERIAL NO.', '', 'ID', '', 'NAME']
    assure_path_exists("StudentDetails/")
    assure_path_exists("TrainingImage/")
    serial = 0
    exists = os.path.isfile("StudentDetails\StudentDetails.csv")
    if exists:
        with open("StudentDetails\StudentDetails.csv", 'r') as csvFile1:
            reader1 = csv.reader(csvFile1)
            for l in reader1:
                serial = serial + 1
        serial = (serial // 2)
        csvFile1.close()
    else:
        with open("StudentDetails\StudentDetails.csv", 'a+') as csvFile1:
            writer = csv.writer(csvFile1)
            writer.writerow(columns)
            serial = 1
        csvFile1.close()
    Id = (txt.get())
    name = (txt2.get())
    if ((name.isalpha()) or (' ' in name)):
        cam = cv2.VideoCapture(0)
        harcascadePath = "haarcascade_frontalface_default.xml"
        detector = cv2.CascadeClassifier(harcascadePath)
        sampleNum = 0
        while (True):
            ret, img = cam.read()
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            faces = detector.detectMultiScale(gray, 1.3, 5)
            for (x, y, w, h) in faces:
                cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
                # incrementing sample number
                sampleNum = sampleNum + 1
                # saving the captured face in the dataset folder TrainingImage
                cv2.imwrite("TrainingImage\ " + name + "." + str(serial) + "." + Id + '.' +
str(sampleNum) + ".jpg",
                            gray[y:y + h, x:x + w])
                # display the frame
                cv2.imshow('Taking Images', img)
            # wait for 100 miliseconds
            if cv2.waitKey(100) & 0xFF == ord('q'):
                break
            # break if the sample number is morethan 100
            elif sampleNum > 100:
                break
        cam.release()
        cv2.destroyAllWindows()
        res = "Images Taken for ID : " + Id
        row = [serial, '', Id, '', name]
        with open('StudentDetails\StudentDetails.csv', 'a+') as csvFile:
            writer = csv.writer(csvFile)
            writer.writerow(row)
        csvFile.close()
        message1.configure(text=res)
    else:
        if (name.isalpha() == False):
            res = "Enter Correct name"
            message.configure(text=res)
```

12) Method for training Faces with HaarCascade and LBPHFaceRecognizer.

```
def TrainImages():
    check_haarcascadefile()
    assure_path_exists("TrainingImageLabel/")
    recognizer = cv2.face_LBPHFaceRecognizer.create()
    harcascadePath = "haarcascade_frontalface_default.xml"
    detector = cv2.CascadeClassifier(harcascadePath)
    faces, ID = getImagesAndLabels("TrainingImage")
    try:
        recognizer.train(faces, np.array(ID))
    except:
        mess._show(title='No Registrations', message='Please Register someone first!!!')
        return
    recognizer.save("TrainingImageLabel\Trainner.yml")
    res = "Profile Saved Successfully"
    message1.configure(text=res)
    message.configure(text='Total Registrations till now  : ' + str(ID[0]))
```

13) Method for displaying Images and labels after detect faces.

```
def getImagesAndLabels(path):
    # get the path of all the files in the folder
    imagePaths = [os.path.join(path, f) for f in os.listdir(path)]
    # create empty face list
    faces = []
    # create empty ID list
    Ids = []
    # now looping through all the image paths and loading the Ids and the images
    for imagePath in imagePaths:
        # loading the image and converting it to gray scale
        pilImage = Image.open(imagePath).convert('L')
        # Now we are converting the PIL image into numpy array
        imageNp = np.array(pilImage, 'uint8')
        # getting the Id from the image
        ID = int(os.path.split(imagePath)[-1].split(".")[1])
        # extract the face from the training image sample
        faces.append(imageNp)
        Ids.append(ID)
    return faces, Ids
```

## Conclusion

Face recognition systems are part of facial image processing applications and their significance as a research area are increasing recently. Implementations of the system are crime prevention, video surveillance, person verification, and similar security activities. The goal is reached by face detection and recognition methods. Knowledge-Based face detection methods are used to find, locate and extract faces in acquired images. Implemented methods are skin color and facial features. Neural network is used for face recognition. RGB color space is used to specify skin color values, and segmentation decreases searching time of face images. Cascade is performed to classify to solve pattern recognition problems since face recognition is a kind of pattern recognition. Classification result is accurate. Classification is also flexible and correct when extracted face image is small oriented, closed eye, and small smiled. Proposed algorithm is capable of detect multiple faces, and performance of system has acceptable good results.

## Future Work

As we see the accuracy of Cascade classifier is roughly 80%. So, we have to increase the accuracy by using SVM (Support Vector Machine). In addition, we can use many cameras in the classroom to capture photo from different angles for students.

## References

1. https://www.edureka.co/blog/python-libraries/

2. https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

3. https://realpython.com/python-gui-tkinter/

4. https://www.w3schools.com/python/numpy_intro.asp

5. https://pandas.pydata.org/

6. https://docs.python.org/3/library/time.html

7. https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.htmll

8. https://iq.opengenus.org/lbph-algorithm-for-face-recognition/

9. https://pythonprogramming.net/facial-recognition-python/?fbclid=IwAR3902s4i5gEzHW2fU3omzQ4NatgxJ4S8XAoLBgTCDI7jMdo669uCNCb5bc

10. https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

11. https://subscription.packtpub.com/book/application_development/9781785280948/1/ch01lvl1sec10/what-can-you-do-with-opencv