Computer Vision

FACIAL RECOGNITION BASED ATTENDANCE SYSTEM USING COMPUTER VISION

Section: **BEE-10**

Submitted To: **Dr. Jameel Nawaz Malik**

Dated: 31st December 2021

Team Members

Name	CMS
MUHAMMAD HAMZA SHAKOOR	**255633**
MUHAMMAD HARIS JAVED	**245021**
MUHAMMAD ABDULLAH SHAHZAD	**262513**
SAAD SALMAN	**243311**

Introduction

We have implemented the algorithms taught in the class. All of these algorithms are directly or indirectly related to computer vision and neural networks and are efficiently implemented in our project. The project will mark the Attendance of specified individuals based on their face recognition. The features of facial expressions are detected and trained, and this will be used to detect them and to mark their Attendance in excel sheets.

Background Information

Face Detection:

The most basic task on Face Recognition is of course, "Face Detecting". Before anything, we need to "capture" a face (Phase 1) in order to recognize it, when compared with a new face captured in (Phase 3).

We used 'Haar' Cascade classifier to detect the faces.

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.

Here we worked with face detection only. 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. OpenCV library of python comes with a trainer as well as a detector. OpenCV already contains many pre-trained classifiers for face, eyes, smile, etc. We used those XML files and we retrained them on our dataset images. (*Files can be download from Haarcascades directory*)

Now we will explain these algorithms.

‘Haar’ Cascades Algorithm:

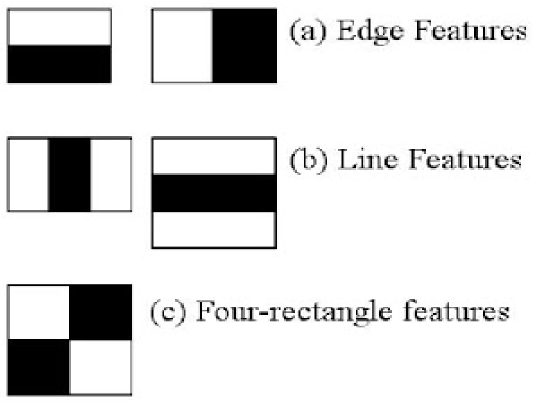
The algorithm can be explained in four stages:

- Calculating ‘Haar’ Features
- Creating Integral Images
- Using Adaboost
- Implementing Cascading Classifiers

It’s important to remember that this algorithm requires a lot of **positive images** of faces and **negative images** of non-faces to train the classifier, similar to other machine learning models.

Calculating ‘Haar’ Features

The first step is to collect the ‘Haar’ features. A **’Haar’ feature** is essentially calculations that are performed on adjacent rectangular regions at a specific location in a detection window. The calculation involves summing the pixel intensities in each region and calculating the differences between the sums. Here are some examples of ‘Haar’ features below.



These features can be difficult to determine for a large image. This is where **integral images** come into play because the number of operations is reduced using the integral image.

Creating Integral Images

Without going into too much of the mathematics behind it (check out the paper if you're interested in that), integral images essentially speed up the calculation of these 'Haar' features. Instead of computing at every pixel, it instead creates sub-rectangles and creates array references for each of those sub-rectangles. These are then used to compute the 'Haar' features.

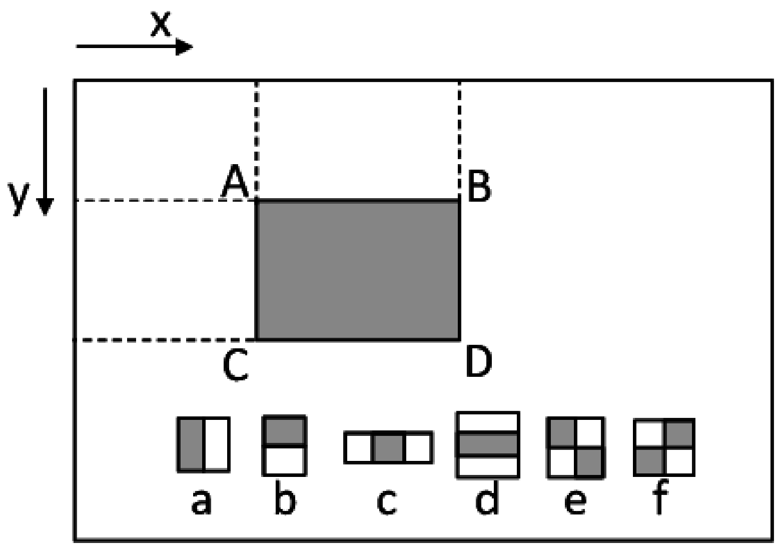


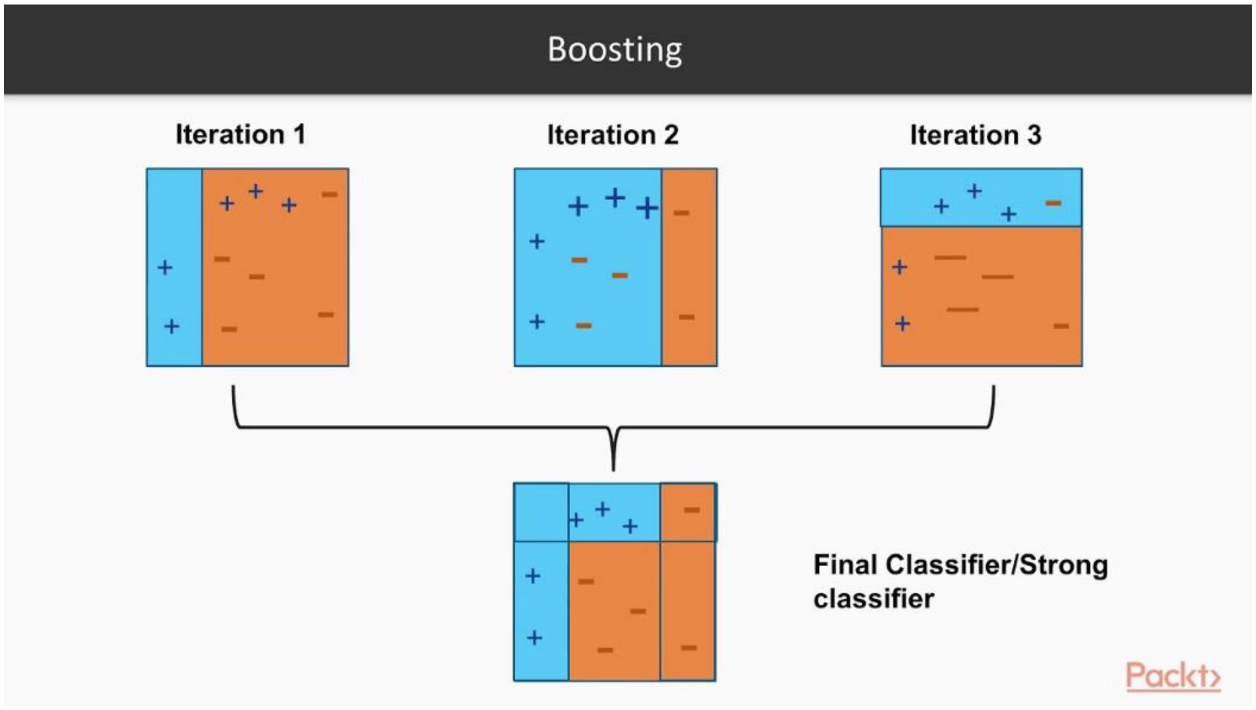
Illustration for how an integral image works.

It’s important to note that nearly all of the ‘Haar’ features will be **irrelevant** when doing object detection, because the only features that are important are those of the object. However, how do we determine the best features that represent an object from the hundreds of thousands of ‘Haar’ features? This is where **Adaboost** comes into play.

Adaboost Training

Adaboost essentially chooses the best features and trains the classifiers to use them. It uses a combination of **“weak classifiers”** to create a **“strong classifier”** that the algorithm can use to detect objects.

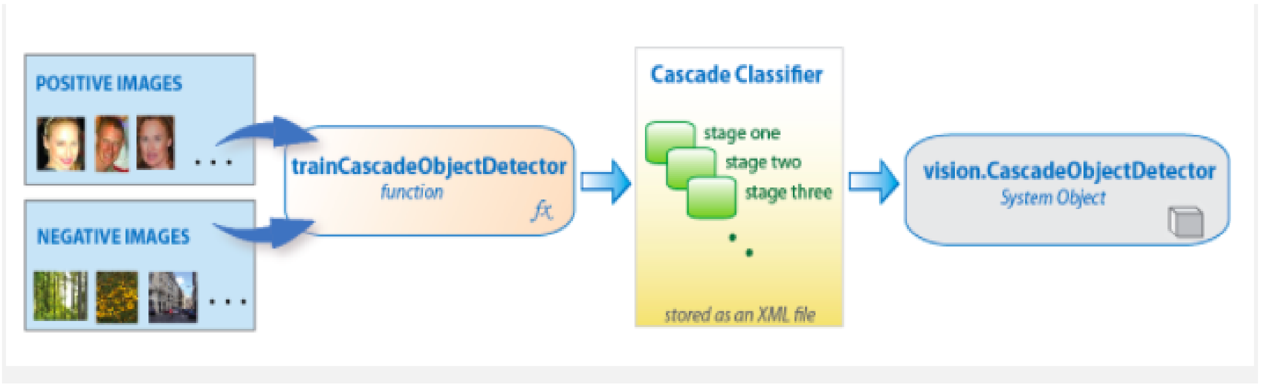
Weak learners are created by moving a window over the input image, and computing ‘Haar’ features for each subsection of the image. This difference is compared to a learned threshold that separates non-objects from objects. Because these are “weak classifiers,” many ‘Haar’ features is needed for accuracy to form a strong classifier.



Representation of a boosting algorithm.

The last step combines these weak learners into a strong learner using **cascading classifiers**.

Implementing Cascading Classifiers



A flowchart of cascade classifiers. (Image Source)

The cascade classifier is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows for a highly accurate classifier from the mean prediction of all weak learners.

Based on this prediction, the classifier either decides to indicate an object was found (positive) or move on to the next region (negative). Stages are designed to reject negative samples as fast as possible, because a majority of the windows do not contain anything of interest.
It's important to maximize a **low false negative rate**, because classifying an object as a non-object will severely impair your object detection algorithm. A video below shows 'Haar' cascades in action. The red boxes denote "positives" from the weak learners. 'Haar' cascades are one of many algorithms that are currently being used for object detection. One thing to note about 'Haar' cascades is that it is very important to **reduce the false negative rate**, so make sure to tune hyperparameters accordingly when training your model.

Face Recognition using LBPH Algorithm

The steps of the Face Recognition using LBPH Algorithm are as follows:

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.

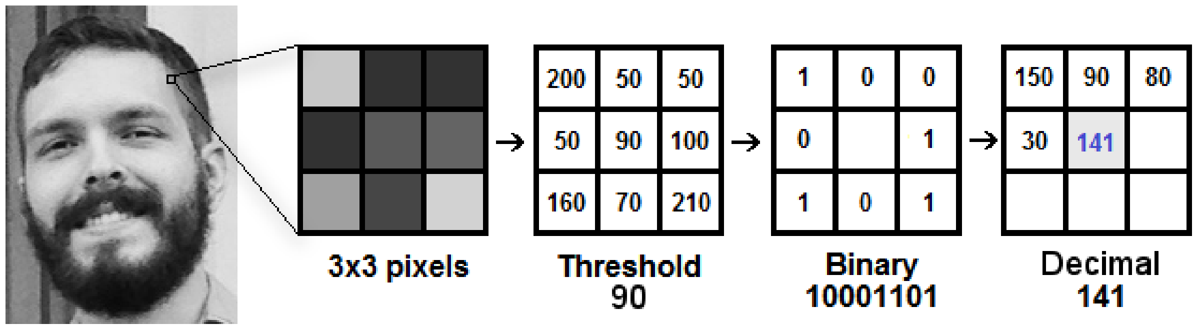
2. Training the Algorithm:

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:

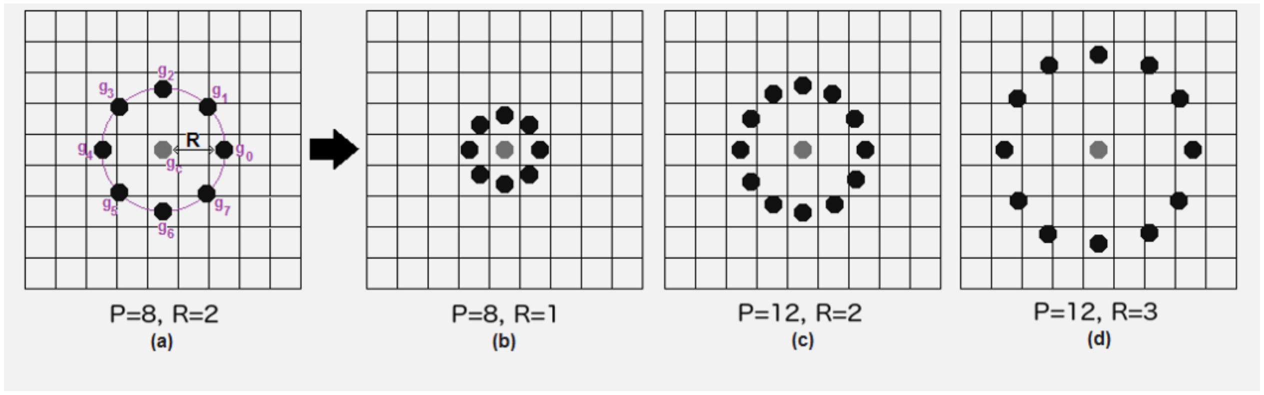
The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameter’s **radius** and **neighbors**.

The image below shows this procedure:



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

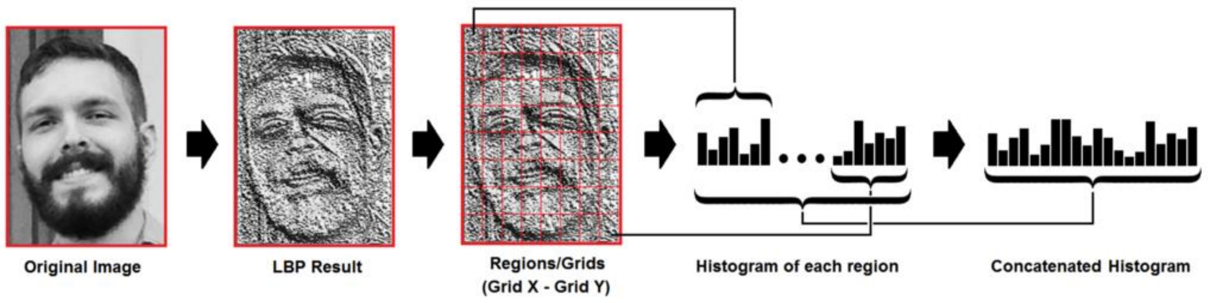
- Suppose we have a facial image in grayscale.
- We can get part of this image as a window of 3x3 pixels.
- It can also be represented as a 3x3 matrix containing the intensity of each pixel (0~255).
- Then, we need to take the central value of the matrix to be used as the threshold.
- This value will be used to define the new values from the 8 neighbors.
- For each neighbor of the central value (threshold), we set a new binary value. We set 1 for values equal or higher than the threshold and 0 for values lower than the threshold.
- Now, the matrix will contain only binary values (ignoring the central value). We need to concatenate each binary value from each position from the matrix line by line into a new binary value (e.g., 10001101). Note: some authors use other approaches to concatenate the binary values (e.g., clockwise direction), but the 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 a pixel from the original image.
- At the end of this procedure (LBP procedure), we have a new image which represents better the characteristics of the original image.
- **Note**: The LBP procedure was expanded to use a different number of radius and neighbors, it is called Circular LBP.



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

4. **Extracting the Histograms**:

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



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

- As we have an image in grayscale, each histogram (from each grid) will contain only 256 positions (0~255) representing the occurrences of each pixel intensity.
- Then, we need to concatenate each histogram to create a new and bigger histogram. Supposing we have 8x8 grids, we will have 8x8x256=16.384 positions in the final histogram. The final histogram represents the characteristics of the image original image.

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.

METHODOLOGY:

There are four phases of the whole program:

1. **Detection of Faces**

   The structure notices the face through a camera in detailed image and captures the image in black and white filter.

2. **Training Stage of Detected Faces**

   In this step, faces are detected from the datasets training after picture accession and pre - processing task. In this step, the recognizer trainer is used to store the histogram values of faces for the training phase.

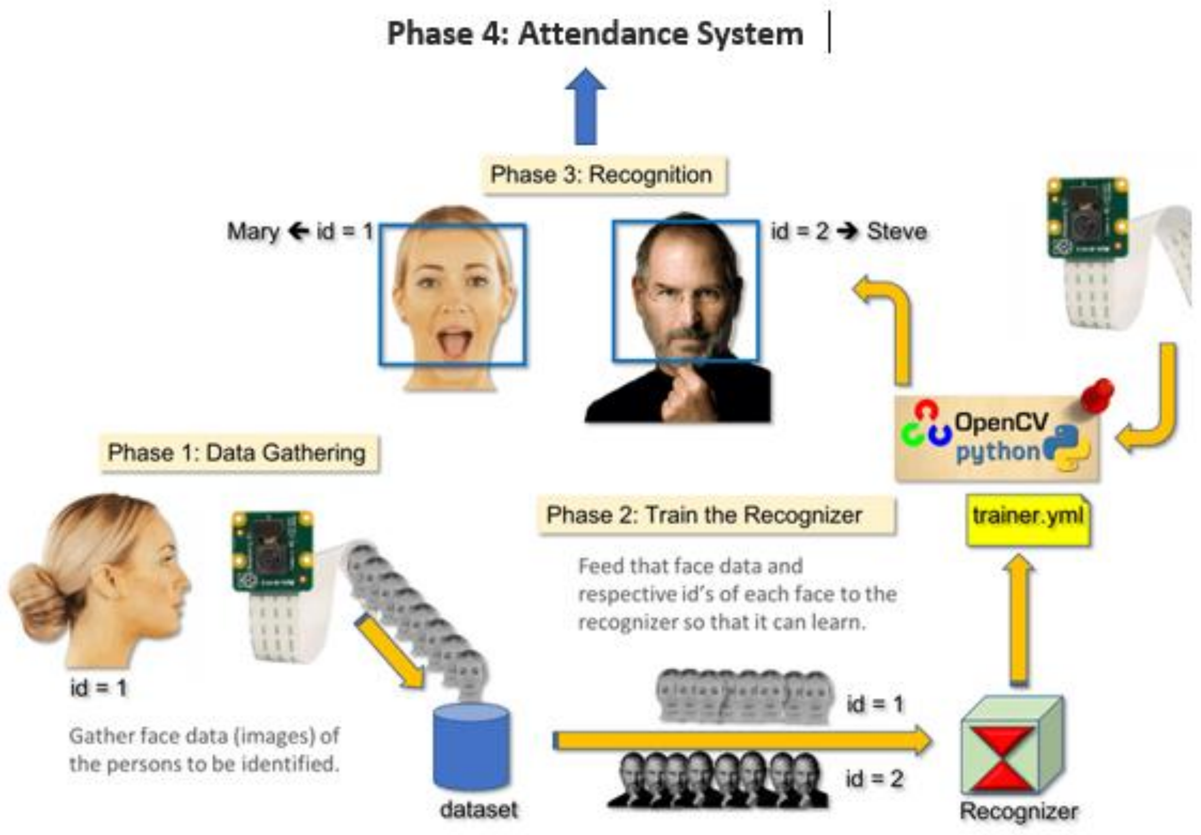
3. **Recognition of detected images**

   Recognizing face images is the final task of face detection part. The 'Haar'-Cascade and Histogram Training Recognizer for Local Binary Patterns will be used to recognize face. The dataset compares the stored images of the face with the images of the input face. The recognized outcome will be shown on the systems screen to check if the face features of input pictures match the database pictures or not.

4. **Attendance system**

   All present students face is captured using live camera, those faces that matched with the trained model, and the faces that cross the 50% confidence are marked as present. Data of present students and absent students is stored in an excel sheet. For each session of the program a new excel file stores the data. In this way attendance of different days or lectures can be taken easily.

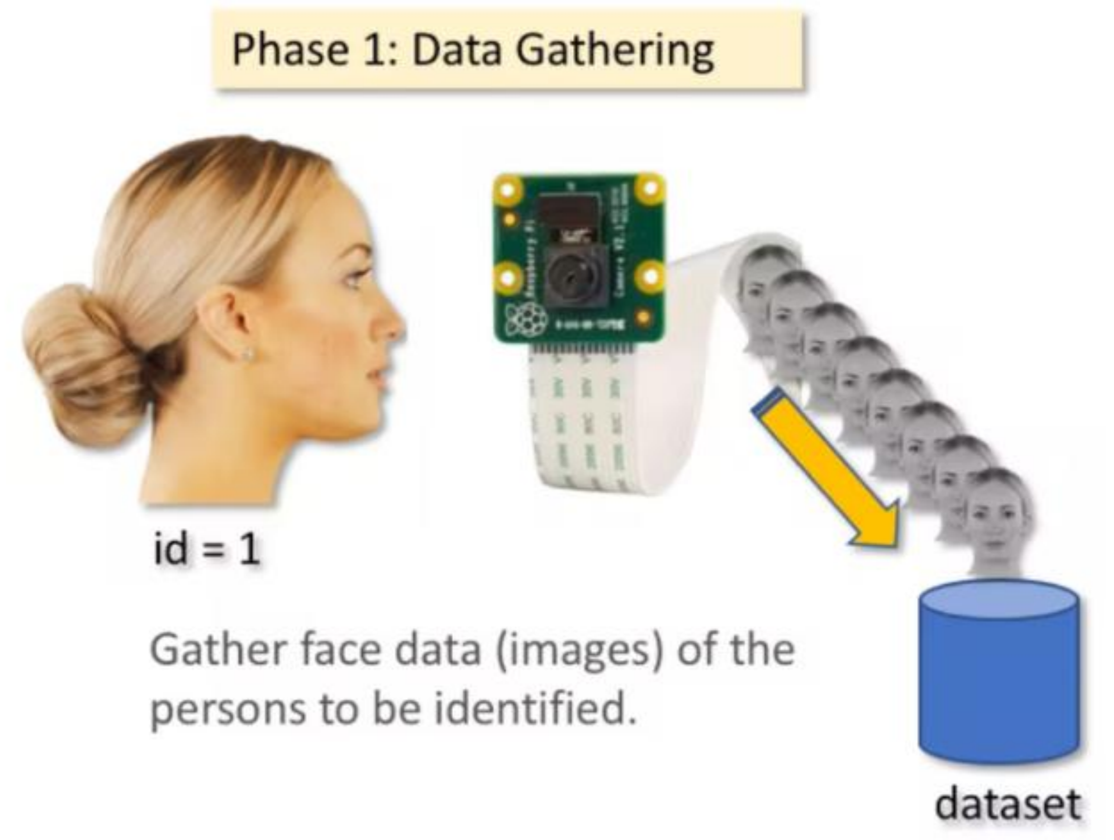
Flow Diagram:


Phase 4: Attendance System
Phase 3: Recognition
Mary ← id = 1
id = 2 → Steve
OpenCV
python
Phase 1: Data Gathering
Phase 2: Train the Recognizer
trainer.yml
Feed that face data and respective id's of each face to the recognizer so that it can learn.
id = 1
Gather face data (images) of the persons to be identified.
dataset
id = 1
id = 2
Recognizer

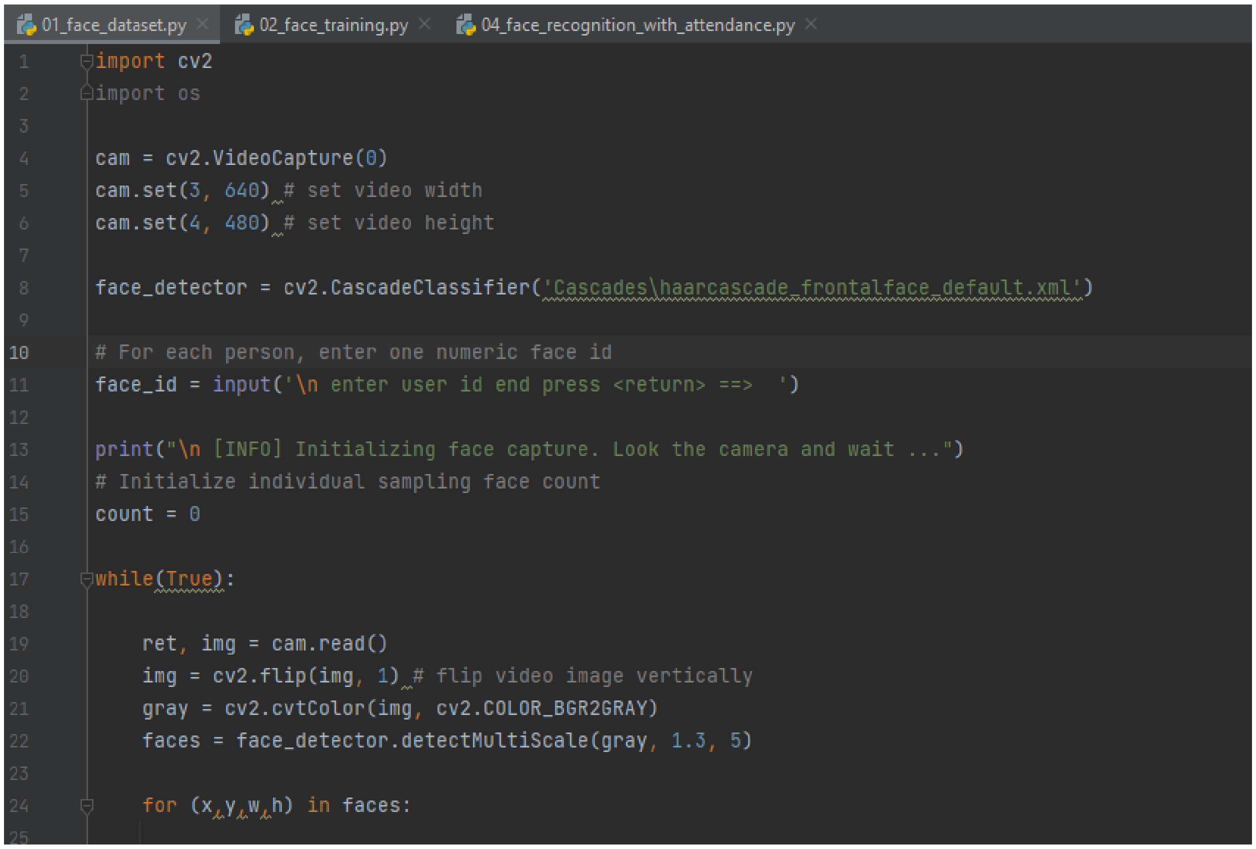
Code Explanation:

There are 3 python files that need to be run in sequence, the code snippets are explained with details below:

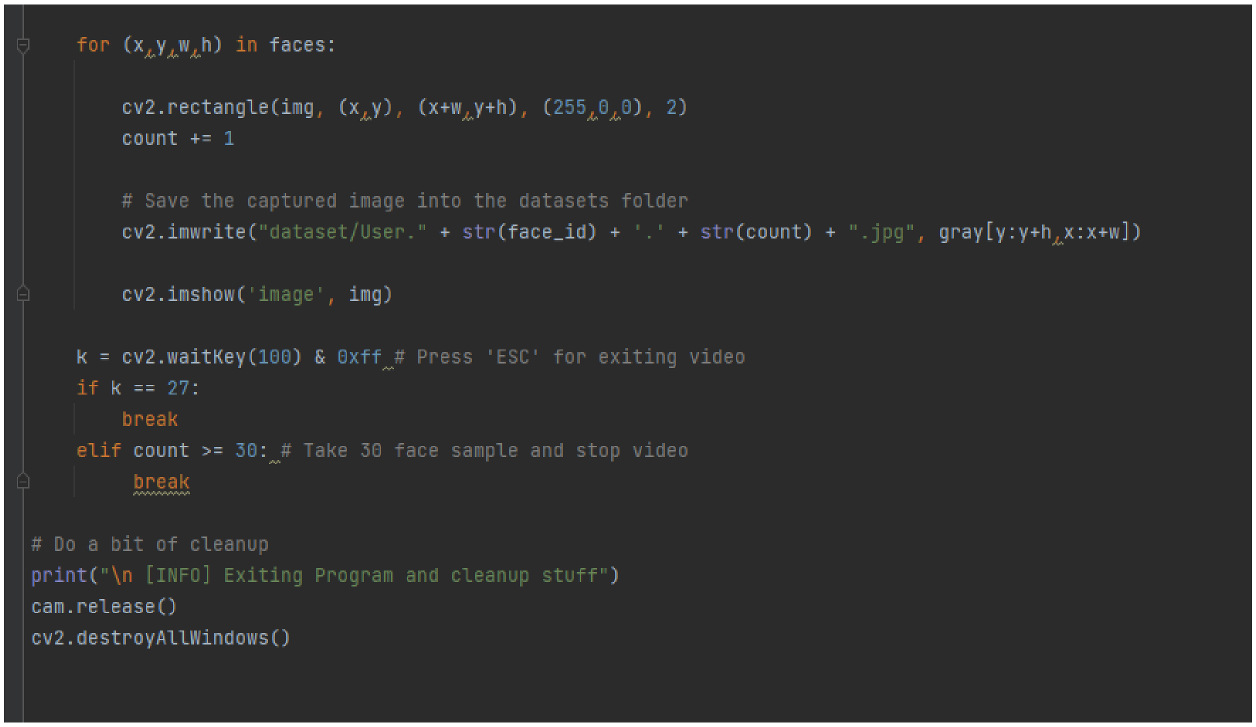
Phase 1: Data Gathering



1. We simply created a dataset, where we stored for each id, a group of photos in gray with the frontal face portion that was used for face detecting.
2. First of all we created a directory, for all the python scripts that we created for our project,
3. Then we saved Pre-made Facial Classifiers that we took from this GitHub link: **https://github.com/Mjrovai/OpenCV-Face-Recognition/tree/master/FacialRecognition**
4. Then we created a folder "dataset" to save the images into it.



```python
import cv2
import os

cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video width
cam.set(4, 480) # set video height

face_detector = cv2.CascadeClassifier('Cascades\haarcascade_frontalface_default.xml')

# For each person, enter one numeric face id
face_id = input('\n enter user id end press <return> ==>  ')

print("\n [INFO] Initializing face capture. Look the camera and wait ...")
# Initialize individual sampling face count
count = 0

while(True):

    ret, img = cam.read()
    img = cv2.flip(img, 1) # flip video image vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
```



```python
    for (x,y,w,h) in faces:

        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        count += 1

        # Save the captured image into the datasets folder
        cv2.imwrite("dataset/User." + str(face_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])

        cv2.imshow('image', img)

    k = cv2.waitKey(100) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break
    elif count >= 30: # Take 30 face sample and stop video
         break

# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

```
faceCascade = cv2.CascadeClassifier('Cascades/'Haar'cascade_frontalface
_default.xml')
```

This is the line that loads the "classifier" (that must be in a directory named "Cascades/", under our project directory)

Then, we set our camera and inside the loop, loaded our input video in grayscale mode. Now we called our classifier function, passing it some very important parameters, as scale factor, number of neighbors and minimum size of the detected face.

```
faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
```

Where, **gray** is the input grayscale image.

scaleFactor is the parameter specifying how much the image size is reduced at each image scale. It is used to create the scale pyramid.

minNeighbors is a parameter specifying how many neighbors each candidate rectangle should have, to retain it. A higher number gives lower false positives.

minSize is the minimum rectangle size to be considered a face.

The function will detect faces on the image. Next, we "mark" the faces in the image, using, for example, a blue rectangle. This is done with this portion of the code:

```
for (x,y,w,h) in faces:
    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]
```

Once we get these locations, we created an "ROI" (drawn rectangle) for the face and present the result with imshow() function.

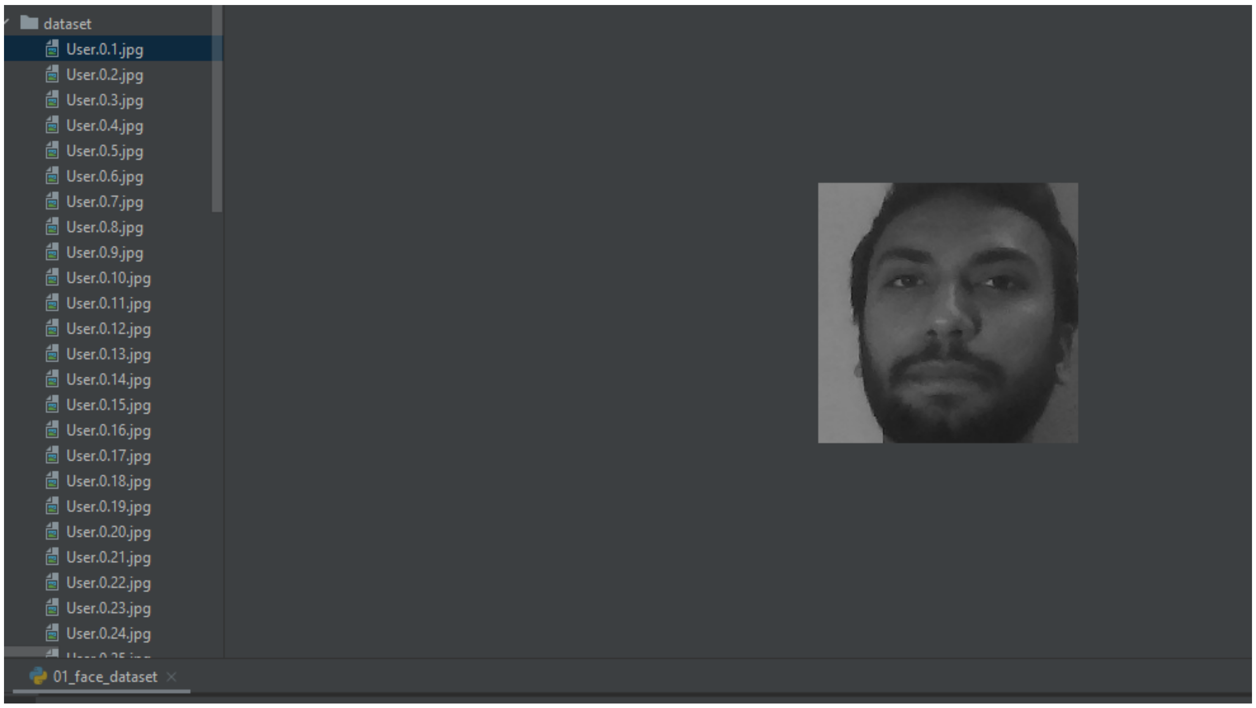
```
face_id = input('\n enter user id end press  ==>  ')
```

then we added, an "input command" to capture a user id, that should be an integer number (1, 2, 3, etc)

And for each one of the captured frames, we should save it as a file on a "dataset" directory:

```
cv2.imwrite("dataset/User." + str(face_id) + '.' + str(count) + ".jpg",
gray[y:y+h,x:x+w])
```

For example, for a user with a face_id = 0, the 1st sample file on dataset/ directory will be something like:

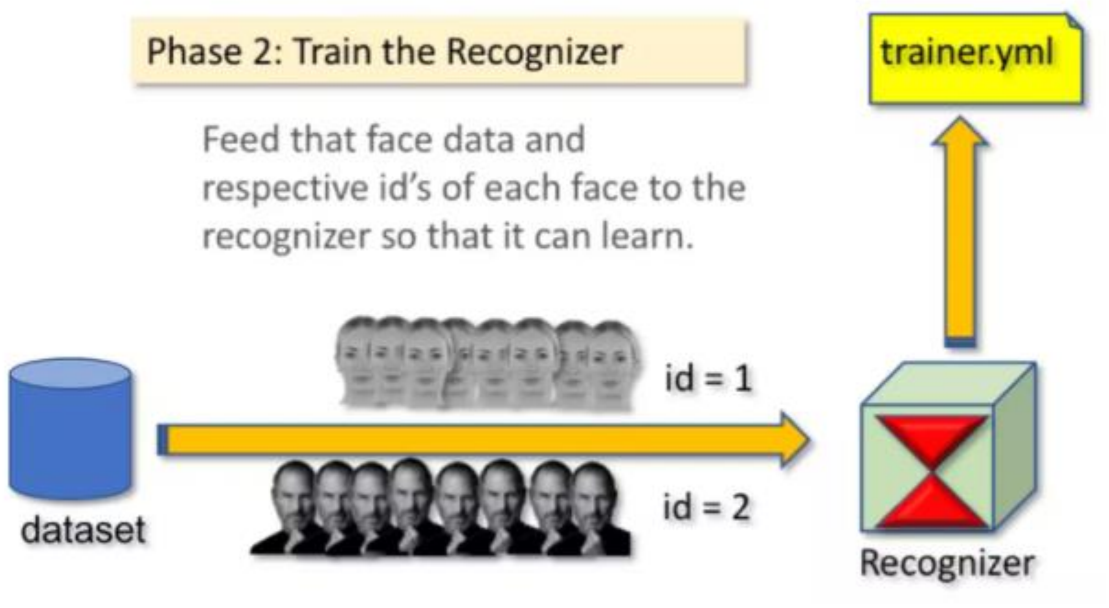


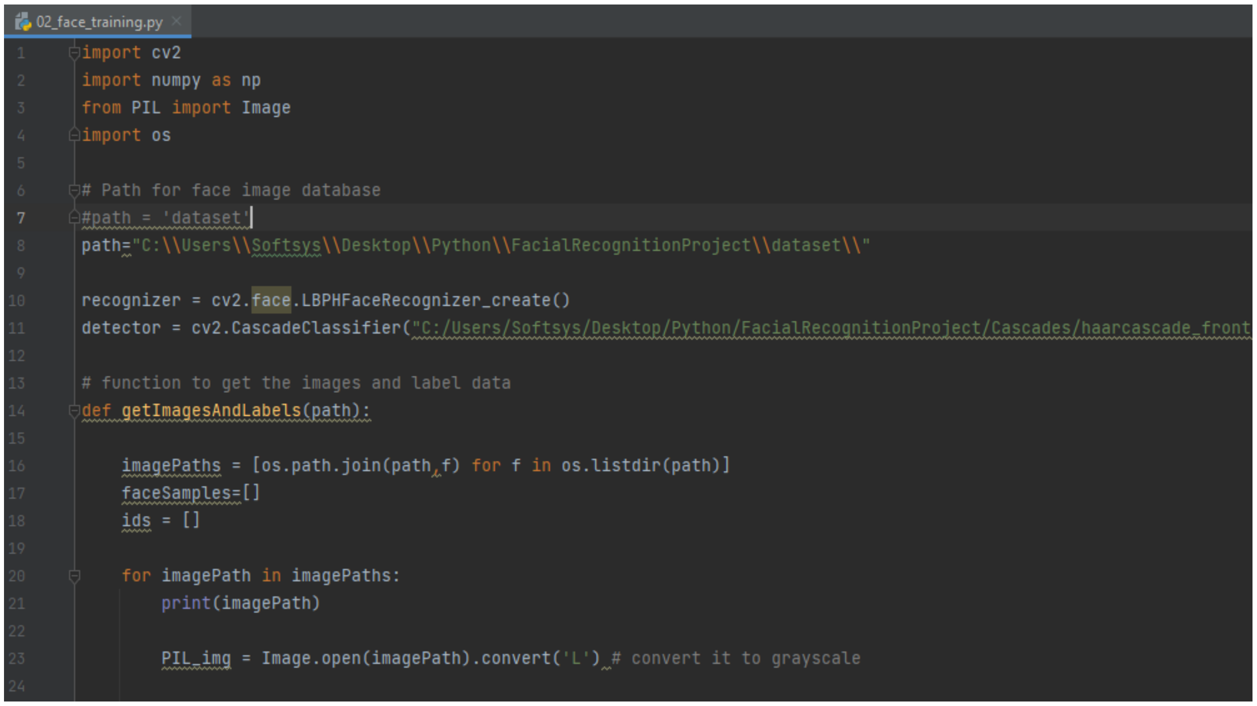
```
dataset
    User.0.1.jpg
    User.0.2.jpg
    User.0.3.jpg
    User.0.4.jpg
    User.0.5.jpg
    User.0.6.jpg
    User.0.7.jpg
    User.0.8.jpg
    User.0.9.jpg
    User.0.10.jpg
    User.0.11.jpg
    User.0.12.jpg
    User.0.13.jpg
    User.0.14.jpg
    User.0.15.jpg
    User.0.16.jpg
    User.0.17.jpg
    User.0.18.jpg
    User.0.19.jpg
    User.0.20.jpg
    User.0.21.jpg
    User.0.22.jpg
    User.0.23.jpg
    User.0.24.jpg
01_face_dataset
```

In our code, we are capturing 30 samples from each id. We can change this number on the last "elif". More samples would increase the accuracy, but it takes more time for gathering, processing, and training.

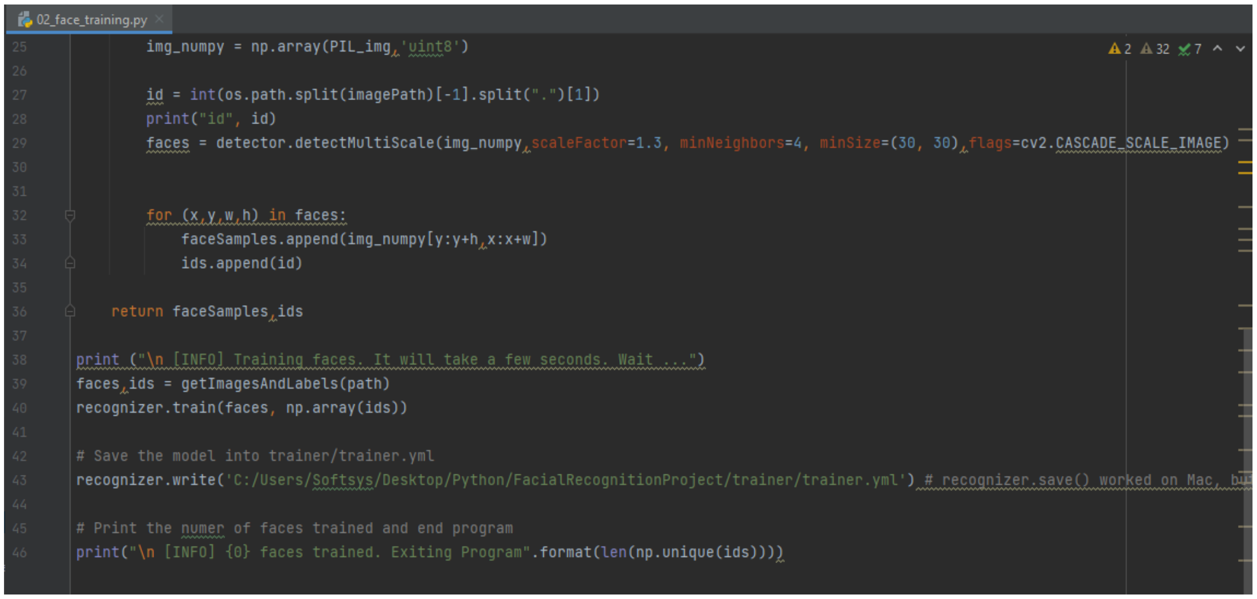
The number of samples is used to break the loop where the face samples are.

Phase 2: Train the Recognizer


Phase 2: Train the Recognizer
trainer.yml
Feed that face data and respective id's of each face to the recognizer so that it can learn.
dataset
id = 1
id = 2
Recognizer



```python
import cv2
import numpy as np
from PIL import Image
import os

# Path for face image database
#path = 'dataset'
path="C:\\Users\\Softsys\\Desktop\\Python\\FacialRecognitionProject\\dataset\\"

recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier("C:/Users/Softsys/Desktop/Python/FacialRecognitionProject/Cascades/haarcascade_front

# function to get the images and label data
def getImagesAndLabels(path):

    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    faceSamples=[]
    ids = []

    for imagePath in imagePaths:
        print(imagePath)

        PIL_img = Image.open(imagePath).convert('L') # convert it to grayscale

```



```python
        img_numpy = np.array(PIL_img,'uint8')

        id = int(os.path.split(imagePath)[-1].split(".")[1])
        print("id", id)
        faces = detector.detectMultiScale(img_numpy,scaleFactor=1.3, minNeighbors=4, minSize=(30, 30),flags=cv2.CASCADE_SCALE_IMAGE)


        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)

    return faceSamples,ids

print ("\n [INFO] Training faces. It will take a few seconds. Wait ...")
faces,ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))

# Save the model into trainer/trainer.yml
recognizer.write('C:/Users/Softsys/Desktop/Python/FacialRecognitionProject/trainer/trainer.yml') # recognizer.save() worked on Mac, bu

# Print the numer of faces trained and end program
print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.unique(ids))))
```

- On this second phase, we took all user data from our dataset and "trainer" the OpenCV Recognizer. This is done directly by a specific OpenCV function. The result would be a .yml file that is saved on a "trainer/" directory
- We used a recognizer, the LBPH (LOCAL BINARY PATTERNS HISTOGRAMS) Face Recognizer, included on OpenCV package. We did this in the following line:

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
```

- The function "getImagesAndLabels (path)", will take all photos on directory: "dataset/", returning 2 arrays: "Ids" and "faces". With those arrays as input, we will "train our recognizer":

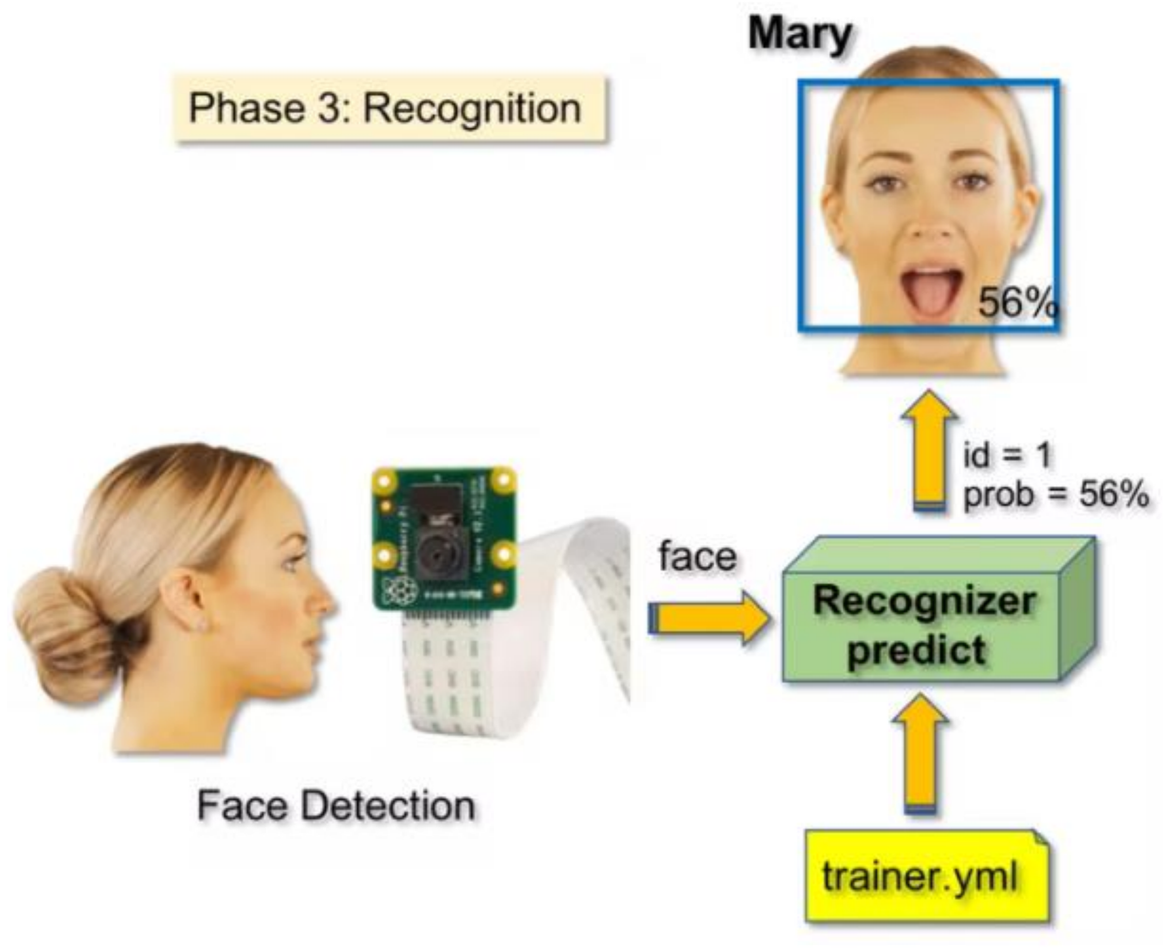
```
recognizer.train(faces, ids)
```

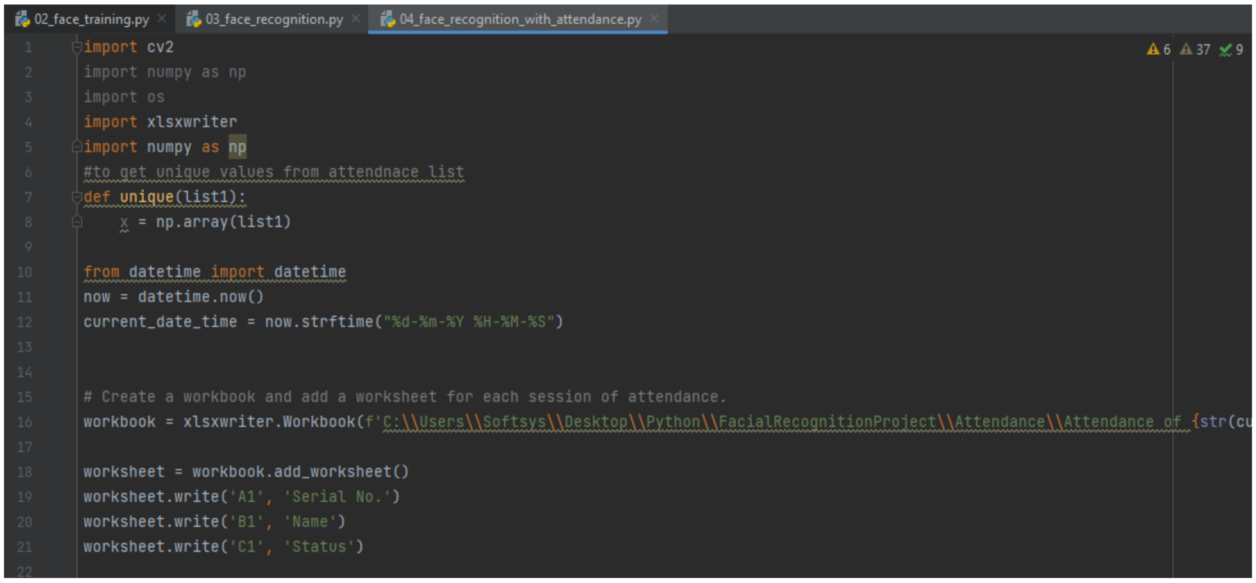
As a result, a file named "trainer.yml" would be saved in the trainer directory that was previously created by us.

Every time, if we perform Phase 1, Phase 2 must also be run.

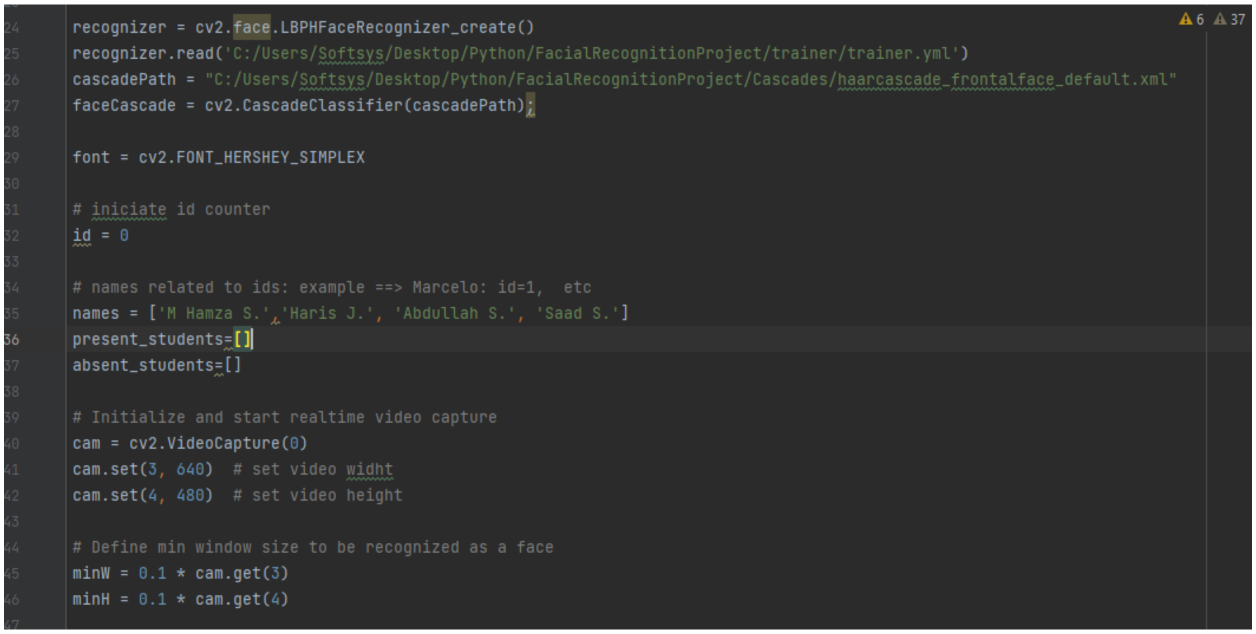
Phase 3: Recognizer

We capture a fresh face on our camera and if this person had his face captured and trained before, our recognizer would make a "prediction" returning its id and an index, shown how confident the recognizer is with this match.

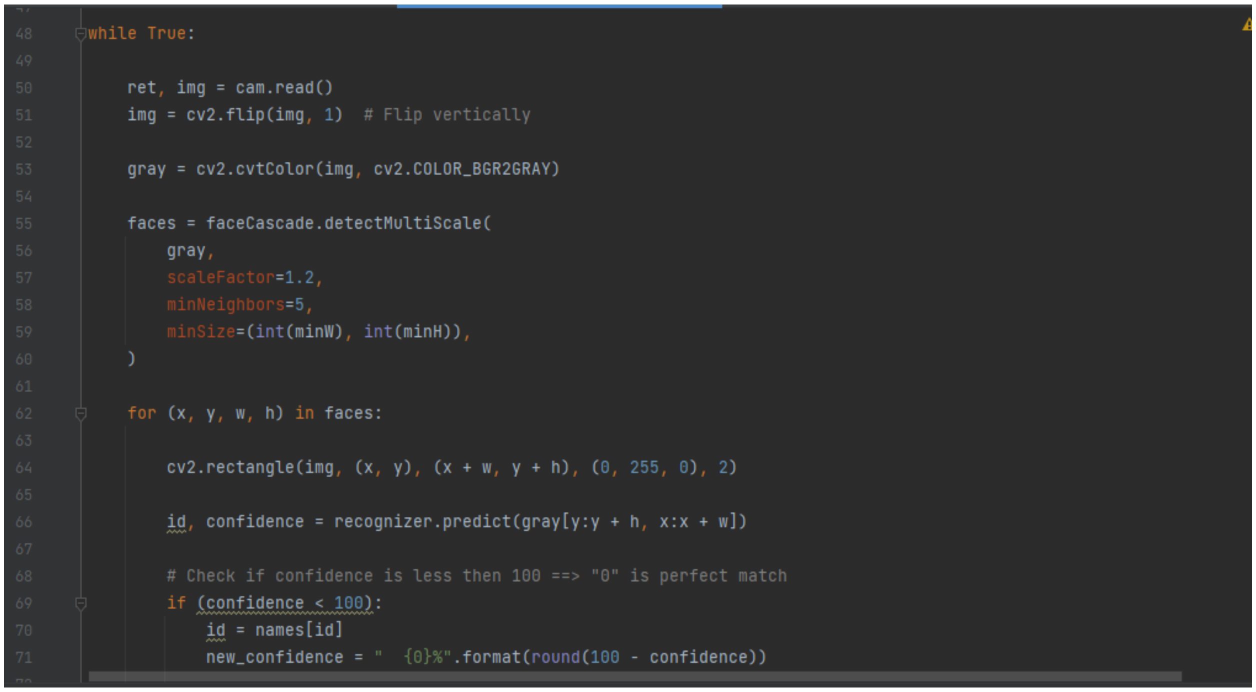




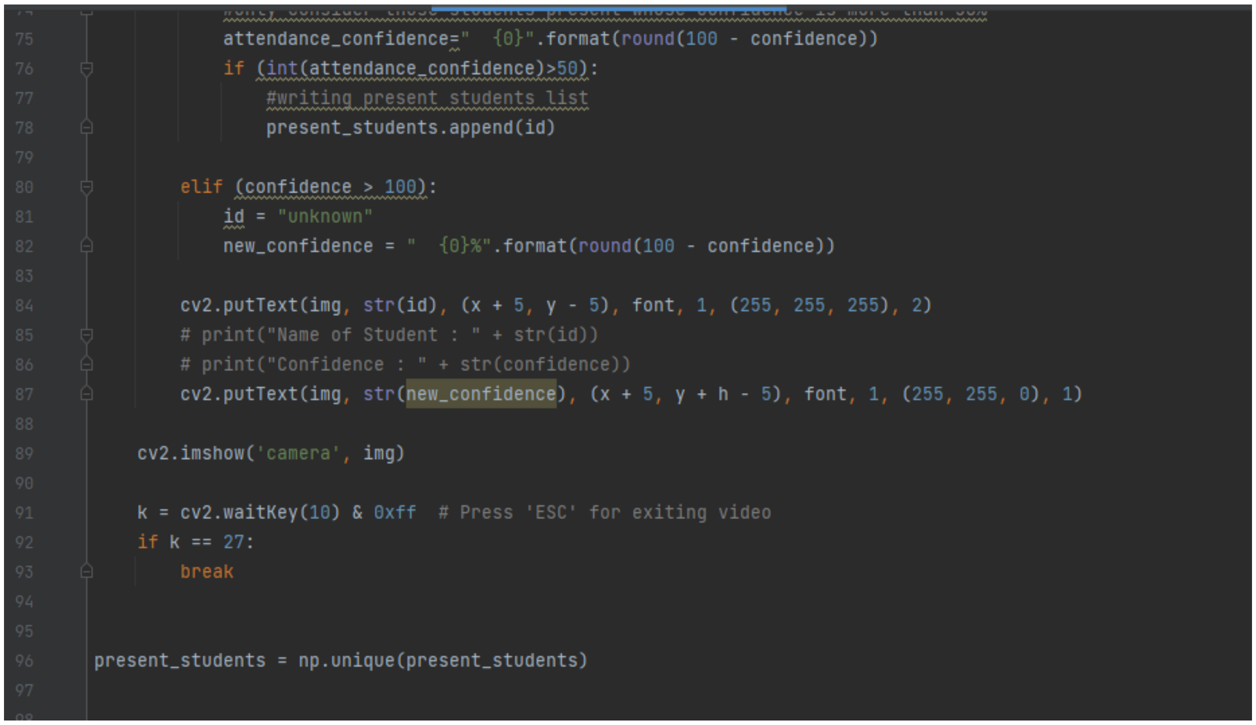
```python
import cv2
import numpy as np
import os
import xlsxwriter
import numpy as np
#to get unique values from attendnace list
def unique(list1):
    x = np.array(list1)

from datetime import datetime
now = datetime.now()
current_date_time = now.strftime("%d-%m-%Y %H-%M-%S")


# Create a workbook and add a worksheet for each session of attendance.
workbook = xlsxwriter.Workbook(f'C:\\Users\\Softsys\\Desktop\\Python\\FacialRecognitionProject\\Attendance\\Attendance of {str(cu

worksheet = workbook.add_worksheet()
worksheet.write('A1', 'Serial No.')
worksheet.write('B1', 'Name')
worksheet.write('C1', 'Status')
```



```python
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('C:/Users/Softsys/Desktop/Python/FacialRecognitionProject/trainer/trainer.yml')
cascadePath = "C:/Users/Softsys/Desktop/Python/FacialRecognitionProject/Cascades/haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);

font = cv2.FONT_HERSHEY_SIMPLEX

# iniciate id counter
id = 0

# names related to ids: example ==> Marcelo: id=1,  etc
names = ['M Hamza S.','Haris J.', 'Abdullah S.', 'Saad S.']
present_students=[]
absent_students=[]

# Initialize and start realtime video capture
cam = cv2.VideoCapture(0)
cam.set(3, 640)  # set video widht
cam.set(4, 480)  # set video height

# Define min window size to be recognized as a face
minW = 0.1 * cam.get(3)
minH = 0.1 * cam.get(4)
```



```python
while True:

    ret, img = cam.read()
    img = cv2.flip(img, 1)  # Flip vertically

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(int(minW), int(minH)),
    )

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

        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

        id, confidence = recognizer.predict(gray[y:y + h, x:x + w])

        # Check if confidence is less then 100 ==> "0" is perfect match
        if (confidence < 100):
            id = names[id]
            new_confidence = "  {0}%".format(round(100 - confidence))
```



```python
            attendance_confidence="  {0}".format(round(100 - confidence))
            if (int(attendance_confidence)>50):
                #writing present students list
                present_students.append(id)

        elif (confidence > 100):
            id = "unknown"
            new_confidence = "  {0}%".format(round(100 - confidence))

        cv2.putText(img, str(id), (x + 5, y - 5), font, 1, (255, 255, 255), 2)
        # print("Name of Student : " + str(id))
        # print("Confidence : " + str(confidence))
        cv2.putText(img, str(new_confidence), (x + 5, y + h - 5), font, 1, (255, 255, 0), 1)

    cv2.imshow('camera', img)

    k = cv2.waitKey(10) & 0xff  # Press 'ESC' for exiting video
    if k == 27:
        break


present_students = np.unique(present_students)
```

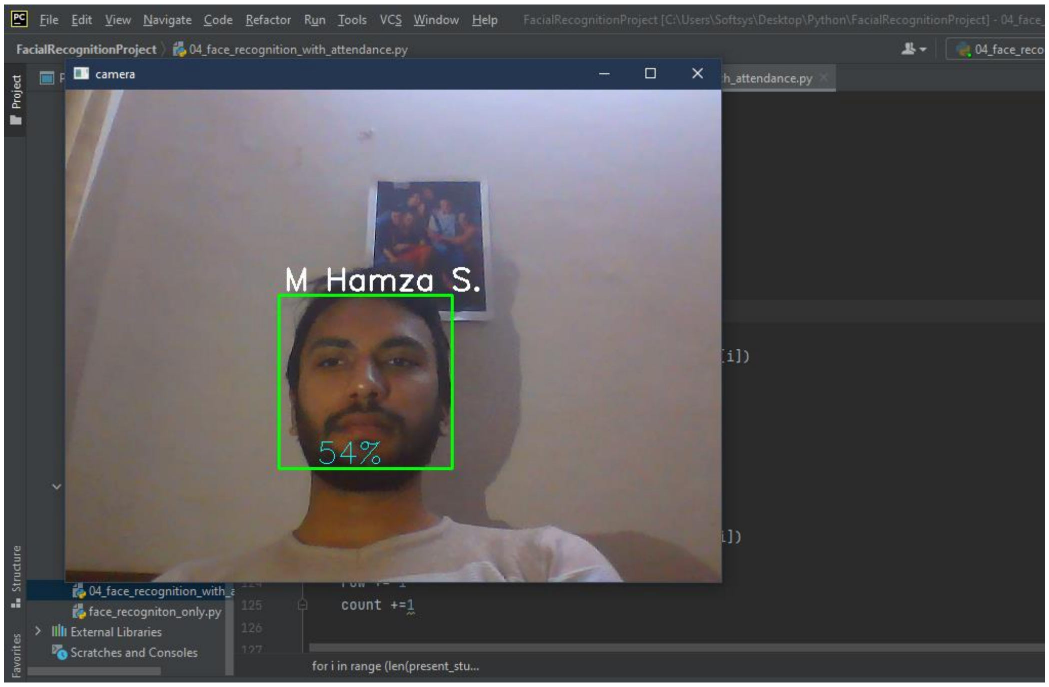
- We are included here a new array, so we displayed "names", instead of numbered ids:

```python
names = ['M Hamza S.','Haris J.', 'Abdullah S.', 'Saad S.']
```

- So, for example:
- M Hamza S with user id=0, Haris J with the user id = 1; Abdulla S user id =2, etc.
- Next, we detected a face, same we did before with the haasCascade classifier. Having a detected face we can call the most important function in the above code:

```
id, confidence = recognizer.predict(gray portion of the face)
```

- The recognizer.predict (), takes as a parameter a captured portion of the face to be analyzed and will return its probable owner, indicating its id and how much confidence the recognizer is in relation with this match.
- Note that the confidence index will return "zero" if it will be considered a perfect match
- And at last, if the recognizer could predict a face, we put a text over the image with the probable id and how much is the "probability" in % that the match is correct ("probability" = 100 - confidence index).
- If not, an "unknow" label is put on the face.

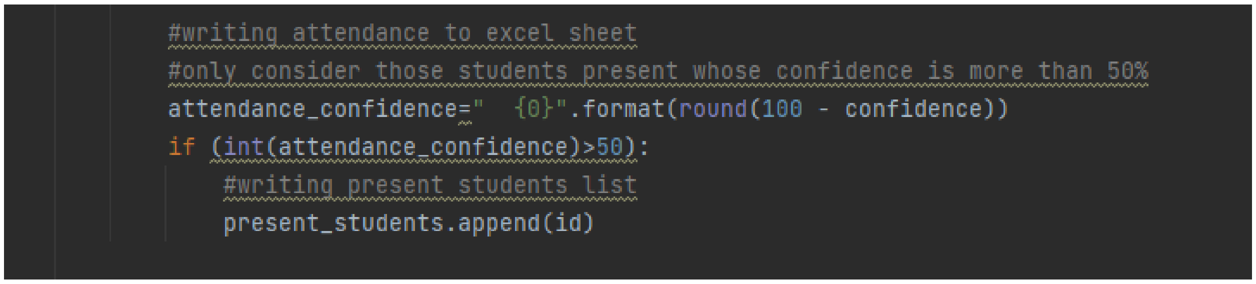


```
count +=1
```

Phase 4: Attendance System

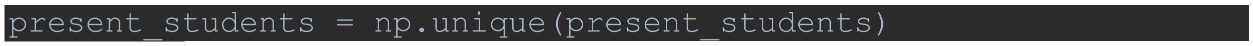
Now we are taking the attendance by using this algorithm:

1. First the attendance confidence is calculated, it’s the probability of the presence of the student.
2. Attendance is counted only when this confidence number is greater than 50 %.
3. It saves the name of the student with respective id in a separate list.



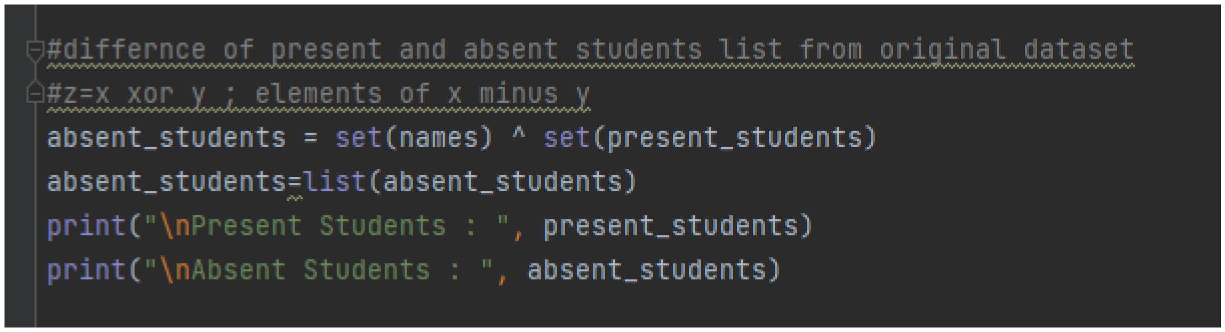
```
#writing attendance to excel sheet
#only consider those students present whose confidence is more than 50%
attendance_confidence="  {0}".format(round(100 - confidence))
if (int(attendance_confidence)>50):
    #writing present students list
    present_students.append(id)
```

4. Then a new list of present students is extracted by using unique function of numpy:



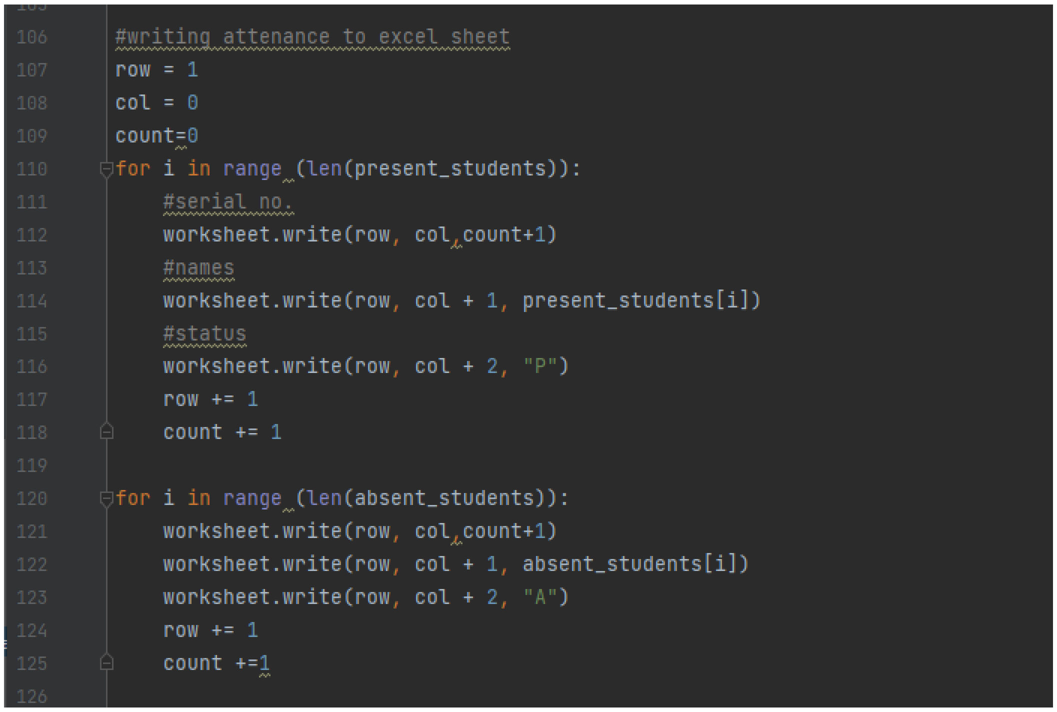
```
present_students = np.unique(present_students)
```

5. After that absent students list is found by using python built in XOR function between names[] list and present_students[] list.



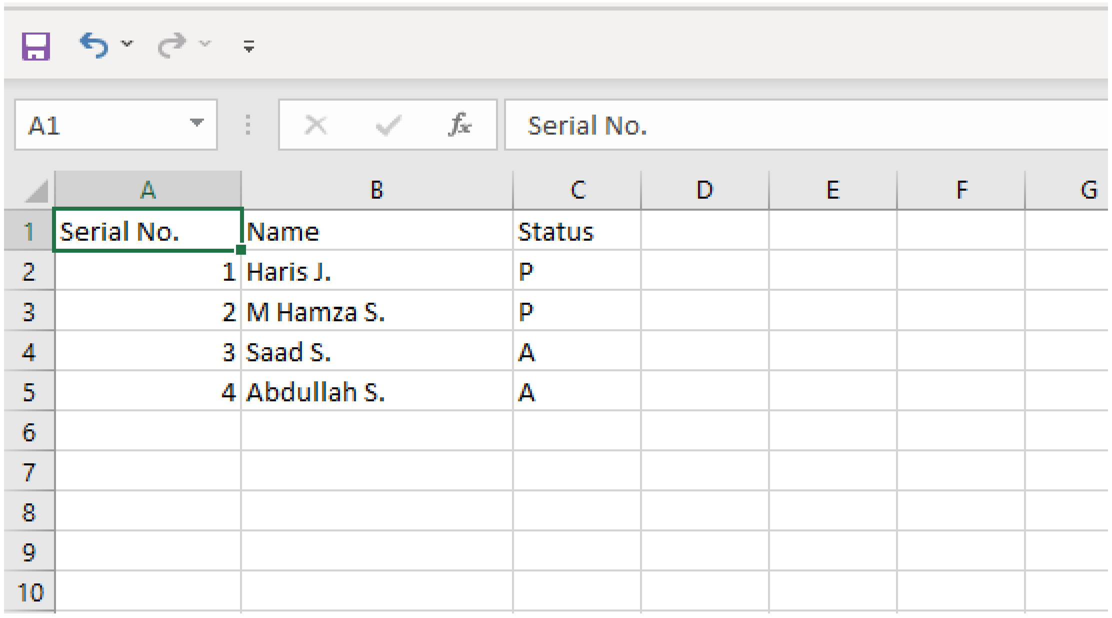
```
#differnce of present and absent students list from original dataset
#z=x xor y ; elements of x minus y
absent_students = set(names) ^ set(present_students)
absent_students=list(absent_students)
print("\nPresent Students : ", present_students)
print("\nAbsent Students : ", absent_students)
```

6. After that both list’s data is stored in excel sheet in three different columns as shown below:



```
#writing attenance to excel sheet
row = 1
col = 0
count=0
for i in range (len(present_students)):
    #serial no.
    worksheet.write(row, col,count+1)
    #names
    worksheet.write(row, col + 1, present_students[i])
    #status
    worksheet.write(row, col + 2, "P")
    row += 1
    count += 1

for i in range (len(absent_students)):
    worksheet.write(row, col,count+1)
    worksheet.write(row, col + 1, absent_students[i])
    worksheet.write(row, col + 2, "A")
    row += 1
    count +=1
```

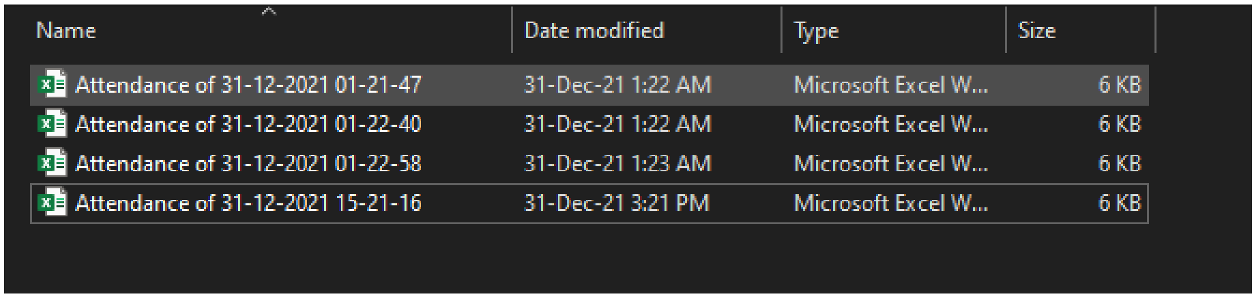
7. Data is stored like this in an excel sheet.



A1 Serial No.

	A	B	C	D	E	F	G
1	Serial No.	Name	Status				
2	1	Haris J.	P				
3	2	M Hamza S.	P				
4	3	Saad S.	A				
5	4	Abdullah S.	A				
6							
7							
8							
9							
10							

8. Attendance will be stored in a new excel file for each session of running the program.
9. In this way we can take attendance at any time and any day, and the program would always save the attendance in a separate file for easy usage and user's best experience.



Name	Date modified	Type	Size
Attendance of 31-12-2021 01-21-47	31-Dec-21 1:22 AM	Microsoft Excel W...	6 KB
Attendance of 31-12-2021 01-22-40	31-Dec-21 1:22 AM	Microsoft Excel W...	6 KB
Attendance of 31-12-2021 01-22-58	31-Dec-21 1:23 AM	Microsoft Excel W...	6 KB
Attendance of 31-12-2021 15-21-16	31-Dec-21 3:21 PM	Microsoft Excel W...	6 KB

Future Advancements:

- Currently the project is written in Python but as we know that Raspberry Pi supports python, we can implement this on the board and create this a full-fledged portable device.

- The classification can be improved by using a better classifier, we used this classifier as our end goal was to make it available for hardware implementation. For hardware small classifier is more efficient in terms of hardware latency (aiding in speedy data gathering, processing and inference).

- There will be certain trade – offs between the speed and accuracy of the project. In the cost analysis we saw that we have made it a speedy one with the trade – off of a bit of accuracy but with the availability of fast hardware like that of FPGA & ASIC's we can implement a big classifier to improve the accuracy as well.

Results/Conclusion:

In this project we incorporated the techniques of Computer Vision and Neural Networks efficiently. We firstly gathered the data using a camera and created a certain dataset for a student containing significant number of images required to train the network. When the dataset of all the students is ready, we use it to train the system. At the inference time the network detects whether the student face is detected or not with a certain confidence level. When the confidence level of a student is greater than 50%, he/she is marked present in the excel sheet which is separate for every session/class. The project greatly helped us to grasp the concepts taught in the class.

References:

https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d

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

https://github.com/opencv/opencv/tree/master/data/haarcascades

https://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html