PROJECT REPORT ON

Biometric image recognition

Submitted by

SURJYA PAUL

SUBHRAJYOTI DAS

SUBHRADEEP SASMAL

SUDIPTA BOSE

WASIM KHAN

For the requirement of the partial fulfillment of Award of degree in

Bachelor of Technology (B.tech) 2019-2023

Under the guidance of

Mr. Sudeep Ghosh

GURU NANAK INSTITUTE OF TECHNOLOGY

Department of Information Technology

157/f, NILGUNJ ROAD, PANIHATI, SODEPUR, KOL-700114

Affiliated to

Maulana Abul Kalam Azad University of Technology

CERTIFICATE

This is to certify that the project work titled **"Biometric Image Recognition"** has been carried out by the following students of Department of Information Technology under my supervision for the partial fulfillment of requirement of Degree of Bachelor of Technology (B.Tech) in Information Technology department of Guru Nanak Institute of Technology of the Maulana Abul Kalam Azad University of Technology during the academic year 2022- 2023.

Surjya Paul	Roll No. : 500419011016
Subhradeep Sasmal	Roll No. : 500419011011
Subhrajyoti Das	Roll No. : 500419011017
Sudipta Bose	Roll No. : 500419011039
Wasim Khan	Roll No. : 500419011054

Project Supervisor
Mr. Sudeep Ghosh
Assistant Professor
(Department of Information Technology)

Head of Department
Dr .Suparna Karmakar
Assistant Professor
(Department of Information Technology)

Invigilator :_____

ACKNOWLEDGEMENT

We are very thankful to our department Hod Dr. Suparna Karmakar for providing us with the project topic of **"Biometric image recognition"** to work on and also thankful to our project mentor Mr. Sudeep Ghosh for his constant guidance without which we wouldn't have been able to complete this project. We are also grateful to all our group members who did the right research about this topic and helped the group in making this project a success. This project has taught us various aspects of machine learning and how we can develop it further. We are looking forward to get such interesting topicsto work on which in turn also develops our knowledge base.

Surjya Paul

Subhradeep Sasmal

Subhrajyoti Das

Sudipta Bose

Wasim khan

TABLE OF CONTENTS

ABSTRACT

Human face recognition is an important part of biometric verification. The methods for utilizing physical properties, such as human face have seen a great change since the emergence of image processing techniques. Human face recognition is widely used for verification purposes, especially if individuals attend to lectures. There is a lot of time lost in classical attendance confirmations. In order to solve this time loss, an Attendance System with Face Recognition has been developed which automatically tracks the attendance status of the students. The Attendance System with Face Recognition performs daily activities of the attendance analysis which is an important aspect of face recognition task. By doing this in an automated manner, it saves time and effort in classrooms and meetings. In the scope of the proposed system, a camera attached to the front of the classroom continuously captures the images of the students, detects the faces in the images, compares them with the database, and thus the participation of the student is determined. Haar filtered AdaBoost is used to detect the real-time human face. Principal Component Analysis (PCA) and Local Binary Pattern Histograms (LBPH) algorithms have been used to identify the faces detected. The paired face is then used to mark course attendance. By using the Attendance System with Facial Recognition, the efficiency of lecture times' utilization will be improved. Additionally, it will be possible to eliminate mistakes on attendance sheets. Face detection and recognition are two of the most popular applications of computer vision. They have many uses in security, biometrics, entertainment, social media, and more. In this blog post, we will give a summary of a face recognition project that we did using open computer vision (OpenCV), a library of programming functions for real-time computer vision. OpenCV provides several algorithms for face detection and recognition, such as Haar-Cascades, and Local Binary Pattern Histograms (LBPH). We used Python as the programming language to implement these algorithms and test their performance on a dataset of face images. The project consisted of four main steps: **Data collection-** We used a webcam to capture images of different people and label them with their names. We stored the images in a folder with subfolders for each person. We also used a text file to store the ID and name of each person. **Face detection-** We used the Haar-Cascade classifier to detect faces in the images. This classifier uses a cascade of simple features to quickly reject non-face regions and focuson the potential face regions. We cropped the detected faces and resized them to a standard size of 640x480 pixels. **Training the dataset-** We used a test set of face images to evaluate the accuracy and speed of each algorithm. We calculated the percentage of correctly recognized faces and the average time taken to recognize one face. **Face recognition-** We used four different algorithms to recognize faces: haarcascade_frontalface_default.xml, haarcascade_eye.xml, haarcascade_smile.xml and LBPH. These algorithms use different techniques to extract features from face images and compare them with a database of known faces. LBPH uses local binary patterns (LBP) to encode the texture information of face images and compute histograms of LBP codes for each image region.

INTRODUCTION

Biometric image recognition is a technique that uses computer vision and machine learning to identify and verify individuals based on their physical characteristics, such as face, fingerprint, iris, palm, or voice. Biometric image recognition has many applications in security, authentication, surveillance, and forensics. However, biometric image recognition also faces many challenges, such as variability of biometric traits, spoofing attacks, privacy concerns, and ethical issues. This paper provides an overview of the current state-of-the-art methods and techniques for biometric image recognition, as well as the main challenges and future directions for research and development in this field.

Face recognition algorithm is a type of computer vision technology that identifies or verifies the identity of an individual by analyzing and comparing patterns in their facial features.

There are various methods and techniques used in face recognition algorithms, but most commonly, they involve the following steps:

Face detection: The first step is to locate the face within an image or video frame. This is usually done using algorithms that identify patterns in the image that match the typical characteristics of a human face.

Feature extraction: Once the face is detected, the algorithm will extract features such as the distance between the eyes, the shape of the nose and mouth, and the overall shape of the face.

Face matching: The extracted features are then compared to a database of known faces to identify or verify the individual's identity. This step typically involves matching the features to a stored template or comparing them to other faces in the database.

There are several types of face recognition algorithms, including Eigenface, Local Binary Patterns, and Deep Learning-based approaches like Convolutional Neural Networks (CNNs)

PROBLEM DEFINITION

Face recognition technology is a biometric system that can identify or verify a person from a digital image or a video source. It works by analyzing the facial features of a person and comparing them with a database of known faces. In this blog post, we will explore some of the advantages of face recognition technology in today's world.

Better security : Face recognition technology can enhance the security of various sectors, such as law enforcement, airports, schools, and workplaces. It can help to track down criminals, find missing people, prevent identity theft, and verify authorized access. For example, face recognition technology can be used to scan the faces of passengers at airports and match them with their passports or visas. This can speed up the security process and reduce the risk of human error.

Improved retail shopping and banking : Face recognition technology can also improve the customer experience and loyalty in retail and banking industries. It can enable personalized marketing, seamless payment, and fraud prevention. For example, face recognition technology can be used to recognize loyal customers and offer them customized discounts or recommendations. It can also be used to verify the identity of customers who want to pay with their faces or withdraw money from ATMs.

More convenience and efficiency : Face recognition technology can also provide more convenience and efficiency for users in various scenarios. It can eliminate the need for passwords, keys, cards, or other forms of identification. It can also automate tasks that require human intervention. For example, face recognition technology can be used to unlock devices, such as smartphones or laptops. It can also be used to tag photos on social media platforms, such as Facebook or Instagram.

Enhanced human-computer interaction : Face recognition technology can also enhance the interaction between humans and computers. It can enable new applications and features that use facial expressions, emotions, gestures, or attention. For example, face recognition technology can be used to support virtual reality and augmented reality applications, such as filters on Snapchat or Instagram. It can also be used to monitor the mood or engagement of students or employees.

Expanded capabilities of devices : Face recognition technology can also expand the capabilities of devices that are equipped with it. It can provide new functions and services that were not possible before. For example, face recognition technology can be used to support artificial intelligence and machine learning capabilities of devices, such as Face ID on iPhone devices from Apple. It can also be used to enable new modes of communication and entertainment, such as video calls or games.

REVIEW WORK

In version 1 we used Haar cascade to identify the face from a Image. Then identify the eye smile . then we used the webcam to identify the face in the fare by collecting real time data from the webcam of the system. Haar cascade is a machine learning technique that can be used for object detection in images and videos. It is based on the idea of using simple features, such as edges and corners, to represent complex objects. The features are organized into a cascade of classifiers, which are trained using a large set of positive and negative examples. The cascade can quickly reject regions that do not contain the object of interest, and focus on the regions that are more likely to contain it. Haar cascade can achieve high accuracy and speed in object detection tasks, such as face detection, eye detection, and license plate recognition. 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 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 the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

In version 2 we user LBPH to train the data set to identify the user and recognize the face. We used LBPH algorithm that was used by python library and OpenCV . To recognize the face .
LBPH stands for Local Binary Pattern Histogram, which is a face recognition algorithm that can recognize faces from different angles and lighting conditions. It works by dividing the image into small regions and computing a binary pattern for each pixel based on its neighbors. Then, it creates a histogram of these patterns for each region and compares them with the histograms of the training images to find the best match. LBPH is known for its simplicity, efficiency and robustness.
For example, if the value 110 appears 50 times a bar like this will be created with this size equal to 50, if 201 appears 110 times and the other bar will be created in this histogram with this size equal to 100. Based on the comparison of the histograms, the algorithm will be able to identify the edges and also the corners of the images. For example, in this first square here, we don't have information about the person's face. So the histogram will be different from this other square that has the border of the face. In short, the algorithm knows which histograms represent borders and which histograms represent the person's main features, such as the colour of the eye, the shape of the mouth, and so on.

In version 3 we supplied the the image which recognized the person from a pregiven image.

In the latest version we are using the webcam to provide live image recognition.

SOFTWARE DEPENDENCIES

- **OpenCV :**

  OpenCV is a library of programming functions that enable computer vision and machine learning applications. It provides a comprehensive set of tools for image processing, feature detection, face recognition, object tracking, video analysis, and more. OpenCV is written in C++ and has bindings for Python, Java, and other languages. It is open source and free for both academic and commercial use. OpenCV is widely used by researchers, developers, and hobbyists around the world.

- **Numpy :**

  NumPy is a Python library that provides a fast and efficient way to work with multidimensional arrays and matrices. NumPy supports various mathematical operations, linear algebra, random number generation, and Fourier transforms. NumPy is the foundation of many other scientific computing packages, such as SciPy, pandas, and scikit-learn. NumPy is widely used in data science, machine learning, engineering, and other fields that require numerical computation.

- **OS :**

  The os module in Python is a built-in module that provides functions for interacting with the operating system. It allows Python to use operating system-dependent functionality in a portable way. The os module includes many functions to work with files and directories, such as creating, renaming, deleting, changing permissions, etc. The os module also provides access to some system information, such as the current working directory, the environment variables, the process ID, etc. The os module can be imported using the statement `import os`.

- **PIL :**

  PIL stands for Python Imaging Library, a powerful and user-friendly library that adds image processing capabilities to your Python interpreter. PIL is a fork of the original PIL by Fredrik Lundh and contributors, and is supported by Tidelift. PIL provides extensive file format support, an efficient internal representation, and fairly powerful image processing features. You can use PIL to load, save, manipulate, and display images in various formats, as well as to work with the individual bands of a multi-band image, such as an RGB image. PIL also offers modules for drawing, enhancing, filtering, math operations, morphing, and more. PIL is designed for fast access to data stored in a few basic pixel formats, and should provide a solid foundation for a general image processing tool.

  - **Image -** PIL (Python Imaging Library) is a popular and powerful library for manipulating images in Python. It provides a wide range of features such as cropping, resizing, rotating, filtering, drawing, and more. PIL can handle many image formats such as JPEG, PNG, GIF, BMP, and TIFF. PIL can also integrate with other libraries such as NumPy and Tkinter to perform more advanced operations. PIL is easy to install and use, and it has a comprehensive documentation and a large community of users and developers.

ALGORITHMS AND DISCUSSION

1. Face, eye and smile detection using OpenCV haarcascade library :-

The face, eye, and smile detection in OpenCV using the haarcascade library is based on the Haar-like feature cascade classifier algorithm. Here's an overview of the algorithms used for each type of detection:

Face Detection:
The face detection algorithm used in OpenCV is based on the Viola-Jones algorithm. It utilizes Haar-like features, which are simple rectangular patterns that capture the contrast difference between adjacent regions of the image. The algorithm scans the image using a sliding window technique, applying the Haar-like features at various scales to detect faces. It uses a trained classifier based on machine learning techniques to classify whether a particular region contains a face or not.

Eye Detection:
The eye detection algorithm is similar to face detection and also utilizes Haar-like features. In this case, the algorithm searches for regions within the detected faces that match the characteristics of eyes. The eye classifier is trained separately to identify the presence of eyes within the face region.

Smile Detection:
Smile detection is also based on Haar-like features and uses a similar approach to face and eye detection. After detecting faces, the algorithm looks for regions within the face where a smile is likely to be present. It uses a trained classifier specific to smile detection to determine if a smile is detected or not.

In all these cases, the Haar-like features are computed efficiently using integral images, and the classifiers are trained on large datasets containing positive and negative examples of the target features.

It's worth mentioning that while Haarcascades are widely used and have been successful for many applications, more advanced techniques like deep learning-based approaches using convolutional neural networks (CNNs) have gained popularity due to their improved accuracy and robustness in various computer vision tasks.

2. Automatic data collection using cv2 :-

Automatic data collection using cv2 (OpenCV) is a technique that allows you to collect data, such as images or videos, from various sources using computer vision capabilities. OpenCV provides a rich set of functions and tools that enable automated data collection for training machine learning models, conducting computer vision research, or performing data analysis.

3. Model training using cv2 , numpy, PIL (image), os :-

Model training using cv2, numpy, PIL (Python Imaging Library), and os is a common approach in computer vision tasks. These libraries provide essential functionalities for data preparation, model training, and evaluation. Here are some key points to consider when using these libraries for model training:

- Data Loading and Preprocessing :

cv2: Use OpenCV to load and preprocess images or videos for training. OpenCV provides functions for reading images in various formats and performing preprocessing tasks such as resizing, cropping, normalization, and data augmentation.
PIL: The PIL library, also known as Pillow, is another powerful image processing library. It allows you to open, manipulate, and save images in different formats. You can use PIL for tasks like resizing, cropping, rotating, and converting image formats.
numpy: The numpy library provides efficient data structures and functions for working with multi-dimensional arrays. It is commonly used for numerical computations and manipulating image data. You can use numpy arrays to store and process image data during training.

- Data Labeling and Annotation:

cv2: OpenCV can be used for manual or automated data labeling and annotation tasks. You can draw bounding boxes, contours, or other shapes on images to mark regions of interest or ground truth labels.
PIL: PIL provides functions to draw on images, add text, or create masks for annotation purposes. These annotations can be useful for training object detection, segmentation, or classification models.
numpy: Numpy arrays can store label information in the form of one-hot encoding or categorical values, depending on the specific training task.

- Dataset Handling and Management:

os: The os module allows you to perform operations on the file system, such as navigating directories, creating new folders, copying files, and listing files in a directory. This is useful for managing your training dataset, organizing images into appropriate directories, or generating file lists for training and validation sets.

- Model Training and Evaluation:

cv2, numpy: These libraries provide the necessary tools for preparing input data and labels in the correct format for training. You can convert images to numpy arrays and preprocess them as required by the model architecture. Additionally, numpy arrays are commonly used for storing model predictions or evaluation metrics.
PIL: PIL can be used to preprocess images before feeding them into the model. You can resize, crop, normalize, or apply other transformations to ensure the input images are in the desired format for model training and evaluation.

4. Face recognition using cv2, numpy, os :-

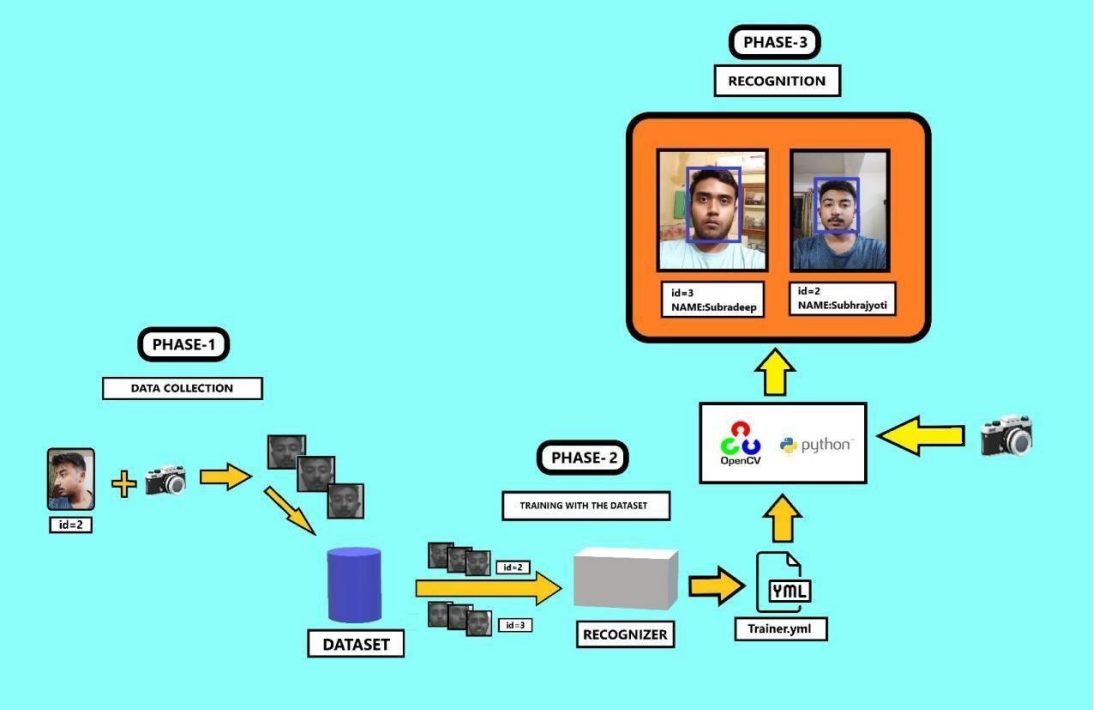
Use os to navigate through directories containing face images and retrieve the file paths. Organize your data into separate folders for different individuals or classes. Load the face images using cv2 or PIL. Convert the images to numpy arrays for further processing. Preprocess the images, such as resizing, normalization, or applying other transformations to ensure consistent input for the face recognition model.

Recognition :

After training the model, you can use it to perform face recognition on new or unseen face images. Detect faces in the input image using face detection algorithms from cv2. Extract features from the detected faces using the same feature extraction techniques used during training. Apply the trained model to compare the extracted features with the known features from the training set. This comparison can be done using distance metrics like Euclidean distance or cosine similarity.
Based on the comparison results, make predictions about the identity of the recognized faces.

PROPOSED SCHEME

- **Block diagram –**



- **Testing the camera -**

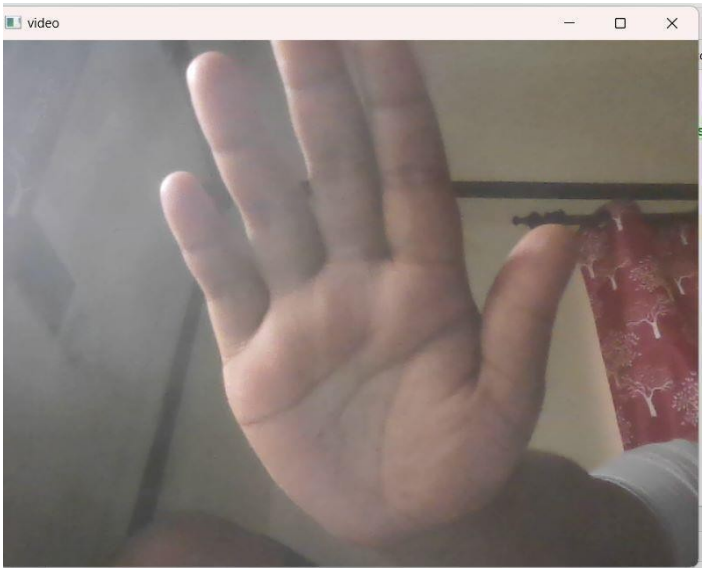
If someone does not have an inbuilt webcam the following can be done :-
At first open your python IDE, then download or install using "pip install opencv" in powershell or command prompt. Once the OpenCV is installed then we need to confirm that the camera is working properly or not by this code :

```
import numpy as np
import cv2

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

while(True) :
    ret, frame = cap.read()
    frame = cv2.flip(frame, -1) # flip camera vertically (if the camera is not flipping don't use it)
    gray= cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame' , frame)
```

```
        cv2.imshow('gray' ,  gray)

        k = cv2.waitKey(30) & 0xff
        if k == 27: # press 'ESC' to quit
            break
cap.release()
cv2.destroyAllWindow()
```

The above code will capture the video string that will be generated by PiCam , displaying both, in BGR color and Gray mode.

*Note : The user must check the camera position and in case of an inbuilt camera no need to use the code.



To finish this code we give this example that our camera works properly

And to end this code we enter the 'ESC' button from keyboard.

Some people found issues while trying to open the camera got the "Assertion failed " error message. That could happen if the camera was not enabled during OpenCV installation and so the camera drive did not install correctly.

So they correct this error using the following command :

'sudo modprobe bcm2835-v4l2' which loads the driver on boot.

- **Face-eye-smile detection –**

The goal of our project is face detection which is working in the backend.
In our project we basically detect the user's face, eyes, smile. So we have downloaded the haarcascade folder of OpenCV library where we pick three files haarcascade_frontalface_default.xml, haarcascade_eye.xml and haarcascade_smile.xml.

1. **Face detection –**
   The most common way to detect someone's face using the user's detection of face. So we are using the file called "haarcascade_frontalface_default.xml" under the haarcascade folder.

   What is haarcascade ?

   It is an Object Detection Algorithm used to identify the faces in an image or a real time video . The algorithm uses edge or line detection features proposed by Viola and Jones in their research paper "Rapid Object Detection using a Boosted Cascade of Simple Features" published in 2001. The algorithm is given a lot of positive images consisting of faces, and a lot of negative images not consisting of any face to train on them.

   The models stored in XML files, and can be read with the OpenCV methods. These include models for face detection, eye detection, upper bodyand lower body detection, license plate detection etc. Below we see some of the concepts proposed by Viola and Jones in their research.

   The GitHub repository is-
   https://github.com/opencv/opencv/tree/master/data/haarcascades

   The requirements for haarcascade: makes sure that the OpenCV library is already installed on your Python IDE

   Working of haarcascade_frontalface_default :

   The Haar-Cascade Face Detection Algorithm is a sliding-window type of algorithm that detects objects based upon its features.

   Haar Face Features
   The Haar-Cascade model, employs different types of feature recognition that include the likes of

- Size and location of certain facial features. To be specific, nose bridge, mouth line and eyes.
- Eye region being darker than upper-cheek region.
- Nose bridge regio being brighter than eye region

This 'XML' file contains a pre-trained model that was created through extensive training and uploaded by Rainer Lienhart on behalf of Intel in 2000.

Rainer's model makes use of the Adaptive Boosting Algorithm (AdaBoost) in order to yield better results and accuracy.

The GitHub repository is:
https://github.com/enespolat25/OpenCV-1-2-3-4/blob/master/haarcascade-frontalface-default.xml

Object detection using haar feature- based cascade classifier is an effective object detection method proposed by Paul Viola and Michael Jones. In this 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 object in other images.

Here, we will work with face detection. Initially, the algorithm needs a lot of positive images(images of faces) and the negative images(images without faces) to train the classifier. Then we need to extract features from it. The good news is that OpenCV comes with a trainer as well as detecter. If we want to train our own classifier of our own object like car, planet, animals etc. we can use OpenCV to create one.

We don’t create our own classifier , cause OpenCV already contains many pretrained classifiers for face, eyes, body, smile etc. those xml file from github repository of haarcascade. The link of the repository is at the page 9.

Now, at first we make .py file named “Face_detection.py”

```
import numpy as np
import cv2

faceCascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)

while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,

        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20,20),
    )

    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[x:x+w, y:y+h]
        roi_color = img[x:x+w, y:y+h]

    cv2.imshow('video',img)

    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

The above few lines of code are all you need to detect a face, using Python and OpenCV.

When you compare with the last code used to test the camera ,you will realize that few parts were added to it. Note the line below :

```
faceCascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
```

This is the line that loads the "classifier (that must be in a directory named "haarcasacades/" under the project directory.

Then we will set our camera and inside the loop we will load our input video in grayscale mode. Now we must call our classifier function passing it some very important parameters as scale factor(scaleFactor) whose size is 1.2 , then number of neighbours (minNeighbour) whose size is 5 and the minimum size (minSize) of detecting face whose size is (20,20).

```
faces = faceCascade.detectMultiScale(
    gray,

    scaleFactor=1.2,
    minNeighbors=5,
    minSize=(20,20),
)
```

Where ,

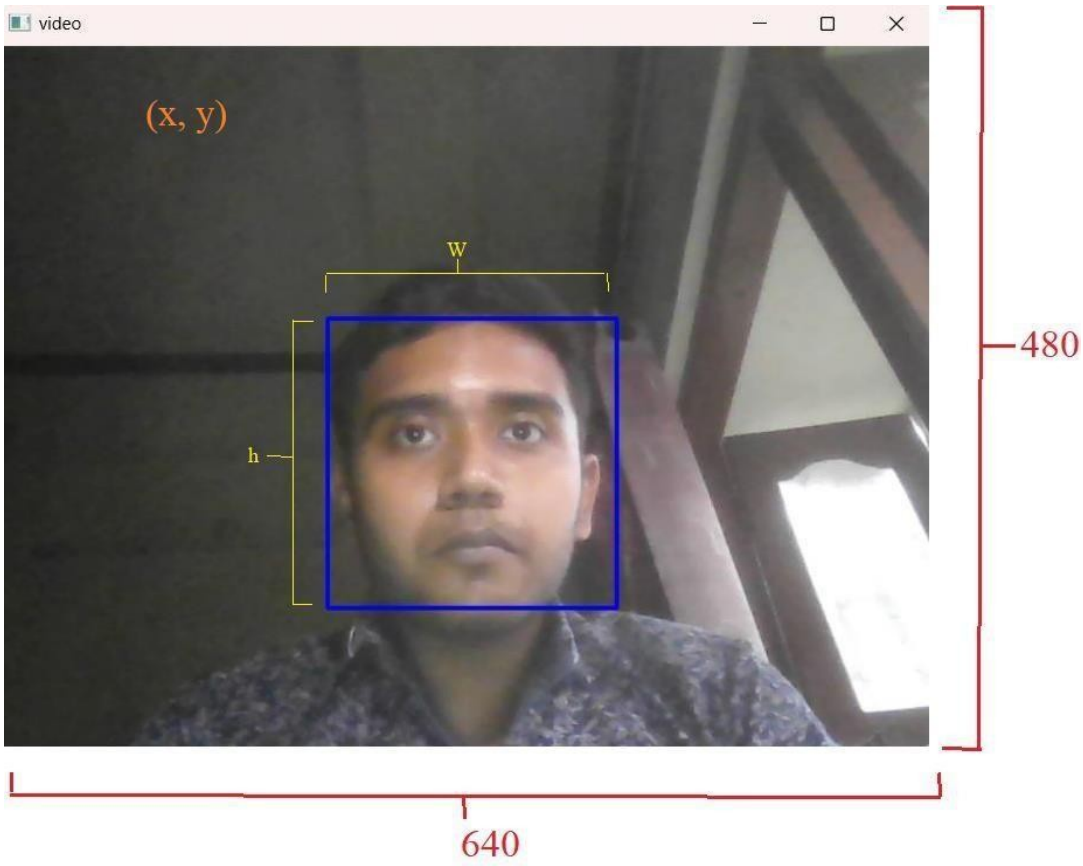
- gray is the input grayscale image
- scaleFactor is the parameter specifying how much the image size is reduced at each image scale.It is used to create the scale pyramid.
- minNeighbors is parameter specifying how many neighbours each candidate rectangle should have , to be retained it. A higher number gives a lower false positive.
- minSize is the minimum rectangle size to be considered as face.

Now,

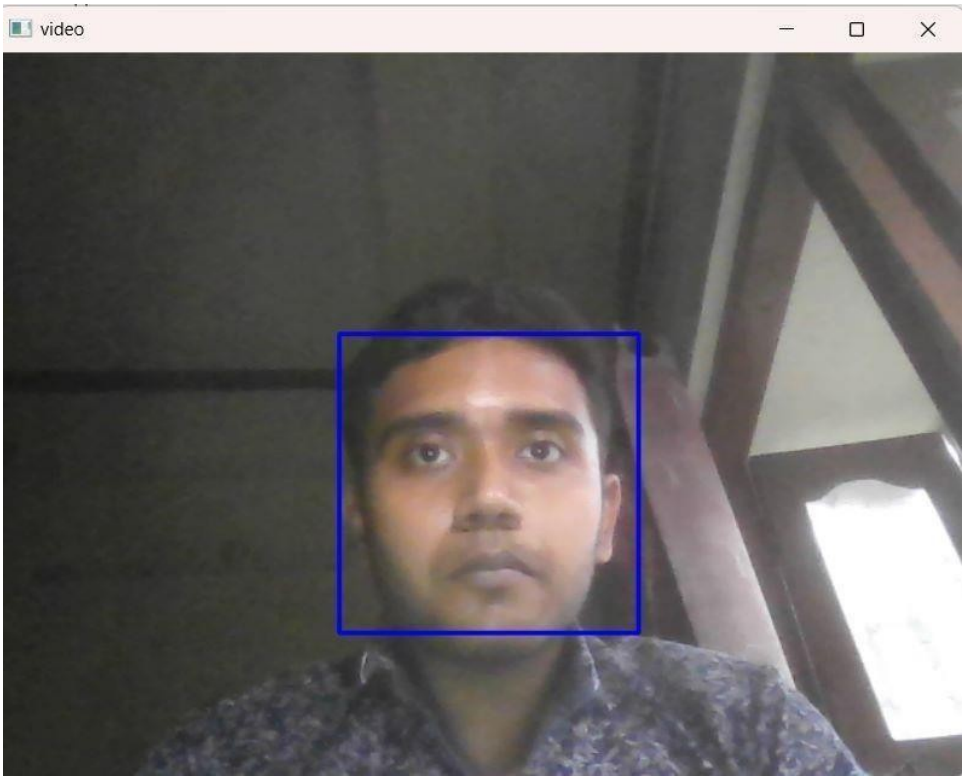
```
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[x:x+w, y:y+h]
    roi_color = img[x:x+w, y:y+h]
```

The function will detect faces on the image. Next we must “mark” the faces in the image, using for example a blue rectangle.

Here if user’s faces are found , it returns the positions of the detected face as a rectangle with the left up corner (x,y) and having “w” as its width and “h” as its height :- (x,y,w,h) which gives this structure



Once we get those locations , we can create an “ROI” (drawn rectangle) or the face and present the result with imshow( ) function. Then we run this python file and get the output



Now we also include classifiers for eye detection and the smile detection. For this case we also include classifier function and rectangle drawn inside the face loop because there would be no sense to detect an eye or smile outside of a face.

2. Eye detection –

The next step is eye detection including the face detection. So, we make a python file named "Face_eye_detection". Here we have used the haarcascade_eye.xml file to detect the eye within the face.

Work of haarcascade_eye :

Haar Cascade is a machine learning-based approach for object detection in images or videos. It is named after the Haar wavelets, a mathematical concept used in image processing.

Haar Cascade Eye detection is a pre-trained algorithm used for detecting human eyes in an image or a video stream. The algorithm works by analyzing the features of the input image and looking for patterns that resemble an eye, such as the round shape and darker area in the center.

The Haar Cascade Eye detection algorithm can be used for various applications such as face recognition, tracking, and even driver drowsiness detection systems. OpenCV, a popular computer vision library, includes pre-trained Haar Cascade classifiers for detecting various objects, including eyes.

Now the face and eye detection will be implemented using the following code :-

```
import numpy as np
import cv2

faceCascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
eyeCascade = cv2.CascadeClassifier('haarcascades/haarcascade_eye.xml')

cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)

while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.3,
        minNeighbors=5,
        minSize=(30,30)
    )

    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[x:x+w, y:y+h]
        roi_color = img[x:x+w, y:y+h]

        eyes = eyeCascade.detectMultiScale(
            roi_gray,
            scaleFactor=1.5,
            minNeighbors=10,
            minSize=(5,5),
        )

        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh),(0,255,0),2)

        cv2.imshow('video', img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

At first we will again call two libraries that we have been called in the face detection python file which is numpy and cv2.

Then we are checking the face using“haarcascade_frontalface_default.xml” file. So this will be done using the following code :

```
faceCascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
```

Then we are checking the eye using the “haarcascade_eye.xml” file :

```
eyeCascade = cv2.CascadeClassifier('haarcascades/haarcascade_eye.xml')
```

Then we set our camera size and position by using the code :

```
cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)
```

where x-axis is 640 and y-axis is 480.

Then we start a while loop where we are checking the face and load our input video in grayscale mode (same as before) by which the loop is being used cause the image can not be still while the input video is running. And again we call our classifier function passing it some very important parameters as scale factor(scaleFactor) whose size is 1.2 , then number of neighbours (minNeighbour) whose size is 5 and the minimum size (minSize) of detecting face whose size is (20,20) and the video is also in grayscale mode .

```
while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.3,
        minNeighbors=5,
        minSize=(30,30)
    )
```

Now we start a for loop where at first we are checking the face and in that we are detecting the eyes :

```
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[x:x+w, y:y+h]
    roi_color = img[x:x+w, y:y+h]

    eyes = eyeCascade.detectMultiScale(
        roi_gray,
        scaleFactor=1.5,
        minNeighbors=10,
        minSize=(5,5),
    )
```

Where x,y,w,h is creating a rectangular box which is in blue colour. So we are use OpenCV and calling a rectangular box where the box will be identifying the face region and the box is build by the width(w), height(h) and adding the x and y axis. So we are adding this part "cv2.rectangle(img,(x,y),(x+w, y+h))" and the "(255,0,0),2" which is the RGB color code of blue color.

And in between this for loop we are now detecting the position of the eyes. So we call some parameters which are scaleFactor, minNeighbors and minSize.

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 parameter specifying how many neighbours each candidate rectangle should have , to be retained it. A higher number gives a lower false positive.

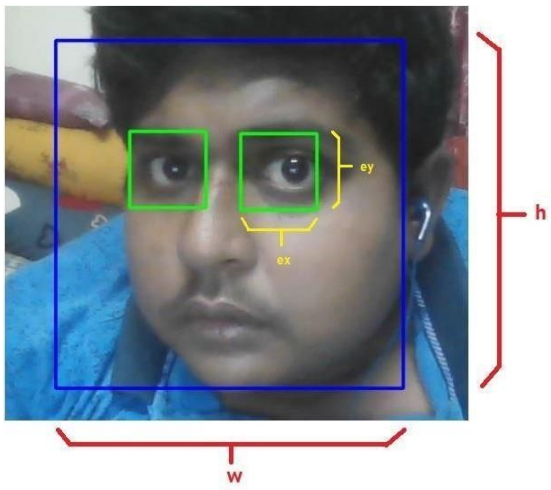
minSize is the minimum rectangle size to be considered as face.

Then we make a rectangular box to detect the position of eyes using the following code :

```
for (ex, ey, ew, eh) in eyes:
    cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh),(0,255,0),2)
```

Where we make a x-axis , y-axis of the eyes in a rectangular box so we are using "ex" and "ey". And "ew" and "eh" is the width and height of the eyes.

Then for making the rectangular box we merge the x-axis with the width and the y-axis with the height and so we use this part "cv2.rectangle(roi_color, (ex,ey), (ex+ew , ey+eh))" and we use the RGB color code which is (0,255,0),2 means the colour code of green color.



Now we are ready to run our code where the machine will detect the user's eyes and face together. So we use this part:

```
    cv2.imshow('video', img)
k = cv2.waitKey(30) & 0xff
if k == 27:
    break

cap.release()
cv2.destroyAllWindows()
```

where,

```
    cv2.imshow('video', img)
k = cv2.waitKey(30) & 0xff
if k == 27:
    break
```

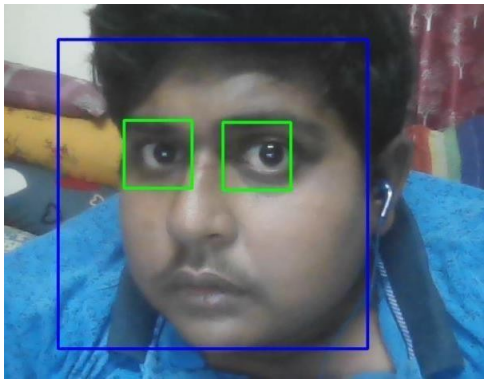
The above part will start the camera and try to detect the face and eyes together and this part

"cap.release"

"cv2.destroyAllWindows( )"

Which is using the keyboard's 'ESC' button to close the running program.

The output is :



3. Face and smile detection –

The next step in the process is to detect the user's face and smile together. So we make a python file and named it "Face_smile_detection". So, we make a python file named "Face_smile_detection". Here we have used "haarcascade_smile.xml" file to detect the smile within the face .

Working of haarcascade_smile :

Haar Cascade is a machine learning-based approach used for object detection. It is a set of Haar-like features that can be trained on positive and negative samples to detect a specific object in an image or video.

In particular, "haarcascade_smile" is a pre-trained cascade classifier that is capable of detecting smiles in images and videos. It is built using the OpenCV (Open Source Computer Vision Library) and has been trained on a large dataset of positive and negative samples to detect smiles with high accuracy.

Using this classifier, one can detect smiles in real-time video streams or in static images, which can be useful for various applications such as emotion recognition, sentiment analysis, and even security systems.

haarcascade_smile.xml is a Haar-cascade classifier that is used to detect features of a face by superimposing predefined patterns over face segments. It is an XML file that can be used with OpenCV for smile detection.

Now the face and smile detection will be implemented by the following code :

```
import numpy as np
import cv2

faceCascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
smileCascade = cv2.CascadeClassifier('haarcascades/haarcascade_smile.xml')

cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)

while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.3,
        minNeighbors=5,
        minSize=(30,30),
    )

    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[x:x+w, y:y+h]
        roi_color = img[x:x+w, y:y+h]

        smile = smileCascade.detectMultiScale(
            roi_gray,
            scaleFactor=1.5,
            minNeighbors=15,
            minSize=(25,25)
        )

        for(xx, yy, ww, hh) in smile:
            cv2.rectangle(roi_color, (xx, yy), (xx + ww, yy + hh),(0.255,0),2)

        cv2.imshow('video', img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

At first we will again import the two libraries numpy and cv2 which we have used in face and eye detection python file.

Then we are checking the face using“haarcascade_frontalface_default.xml” file. So this will be done using the following code :

```
faceCascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
```

Then we are checking the smile using the “haarcascade_smile.xml” file :

```
smileCascade = cv2.CascadeClassifier('haarcascades/haarcascade_smile.xml')
```

Then we set our camera size and position by using the code :

```
cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)
```

where x-axis is 640 and y-axis is 480.

Then we start a while loop where we are checking the face and load our input video in grayscale mode (same as before) by which the loop is being used cause the image can not be still while the input video is running. And again we call our classifier function passing it some very important parameters as scale factor(scaleFactor) whose size is 1.2 , then number of neighbours (minNeighbour) whose size is 5 and the minimum size (minSize) of detecting face whose size is (20,20) and the video is also in grayscale mode .

```
while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.3,
        minNeighbors=5,
        minSize=(30,30)
    )
```

Now we start a for loop where at first we are checking the face and in that we are detecting the smile :

```
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[x:x+w, y:y+h]
    roi_color = img[x:x+w, y:y+h]

    smile = smileCascade.detectMultiScale(
        roi_gray,
        scaleFactor=1.5,
        minNeighbors=15,
        minSize=(25,25)
    )
```

Where x,y,w,h is creating a rectangular box which is in blue colour. So we are use OpenCV and calling a rectangular box where the box will be identifying the face region and the box is build by the width(w), height(h) and adding the x and y axis. So we are adding this part "cv2.rectangle(img,(x,y),(x+w, y+h))" and the "(255,0,0),2" which is the RGB color code of blue color.

And in between this for loop we are now detecting the position of the smile. So we call some parameters which are scaleFactor, minNeighbors and minSize.

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 parameter specifying how many neighbours each candidate rectangle should have , to be retained it. A higher number gives a lower false positive.

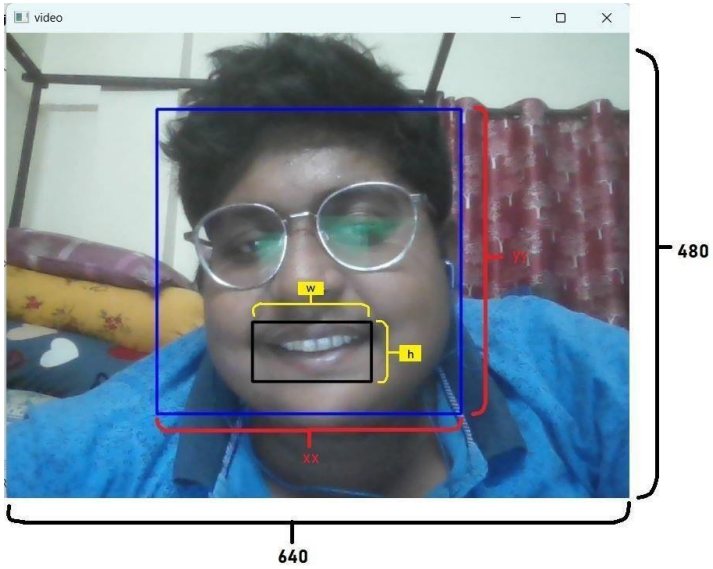
minSize is the minimum rectangle size to be considered as face.

Then we make a rectangular box to detect the position of smile using the following code :

```
    for(xx, yy, ww, hh) in smile:
        cv2.rectangle(roi_color, (xx, yy), (xx + ww, yy + hh),(0.255,0),2)
```

Where we make a x-axis , y-axis of the smile in a rectangular box so we are using "xx" and "yy". And "ww" and "hh" is the width and height of the smile.

Then for making the rectangular box we merge the x-axis with the width and the y-axis with the height and so we use this part “cv2.rectangle(roi_color, (xx,yy), (xx+ww , yy+hh))” and we use the RGB color code which is (0.255,0),2 means the colour code of black color.



Now we are ready to run our code where the machine will detect the user’s smile and face together. So we use this part:

```
        cv2.imshow('video', img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

where,

```
        cv2.imshow('video', img)
k = cv2.waitKey(30) & 0xff
if k == 27:
    break
```

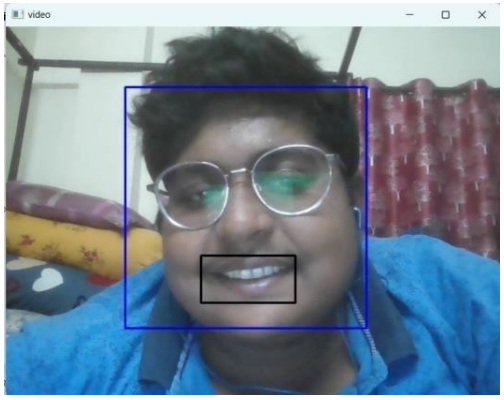
The above part will start the camera and try to detect the face and smile together and this part

“cap.release”

“cv2.destroyAllWindows( )”

Which is using the keyboard’s ‘ESC’ button to close the running program.

The output is :



4. Face , eye and smile detection –

Now our aim is to detect the face, smile and eyes of the user together. So we make a python file and named it “Face_eye_smile_detection”. So at first we are calling the libraries numpy and cv2 then we put the classifiers using opencv which are “haarcascade_frontalface_default.xml” , “haarcascade_eye.xml” , “haarcascade_smile.xml” which is putting in three respective parameters which are faceCascade , eyeCascade and smileCascade.

```
faceCascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
eyeCascade = cv2.CascadeClassifier('haarcascades/haarcascade_eye.xml')
smileCascade = cv2.CascadeClassifier('haarcascades/haarcascade_smile.xml')
```

Now, these classifiers are helping us to detect the face, eye and smile of the user. Then we need to start our camera again so that we put this code :

```
cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)
```

where, x-axis is 640 and y-axis is 480.

Then we are starting a while loop where we are checking the face and load our input video in grayscale mode by which the loop is being used cause the image cannot be still while , when the

output is showing in video mode and again we are calling our classifier function passing it some very important parameters as scale factor (scaleFactor) whose size is 1.3 , then the number of neighbors (minNeighbors) whose size is 5 and the minimum size (minSize) of detecting face whose size is (30,30) and the video is also in gray mode.

```
while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.3,
        minNeighbors=5,
        minSize=(30,30)
    )
```

Now we are starting a for loop where we are checking the face , eyes and the smile together. So at first we make a for loop detection of x y axis and height and width and making a rectangle and in between this we try to find the eyes. We are calling our classifier function passing it some very important parameters as scale factor (scaleFactor) whose size is 1.5 , then the number of neighbors (minNeighbors) whose size is 5 and the minimum size (minSize) of detecting face whose size is (5,5) and the video is also in gray mode.

So we are using this part :

```
for(x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[x:x+w, y:y+h]
    roi_color = img[x:x+w, y:y+h]

    eyes = eyeCascade.detectMultiScale(
        roi_gray,
        scaleFactor=1.5,
        minNeighbors=5,
        minSize=(5,5),
    )
```

And then our aim is to get the smile in between it so at first we call the rectangle of the eyes in a for loop and in between it we call the smile with the parameters which are roi_gray then the

scale factor (Factor) whose size is 1.5 , then the number of neighbors (minNeighbors) whose size is 15 and the minimum size (minSize) of detecting face whose size is (25,25).

```
    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0,255,0),2)

    smile = smileCascade.detectMultiScale(
        roi_gray,
        scaleFactor=1.5,
        minNeighbors=15,
        minSize=(25,25)
    )
```

Then we call the smile rectangle color with RGB color which is black and also with the height and width with x and y axis so we use this part :

```
for (xx, yy, ww, hh) in smile:
    cv2.rectangle(roi_color, (xx, yy), (xx+ww, yy+hh), (0.255,0),2)
```

where "(0.255,0),2" is RGB color code of black and "xx+ww" , "yy + hh" which makes the rectangle of smile.

Now we are ready to run our code where the machine will detect the user's smile , eyes with face together. So we use this part:

```
    cv2.imshow('video', img)
k = cv2.waitKey(30) & 0xff
if k == 27:
    break

cap.release()
cv2.destroyAllWindows()
```

where,

```
        cv2.imshow('video', img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
```

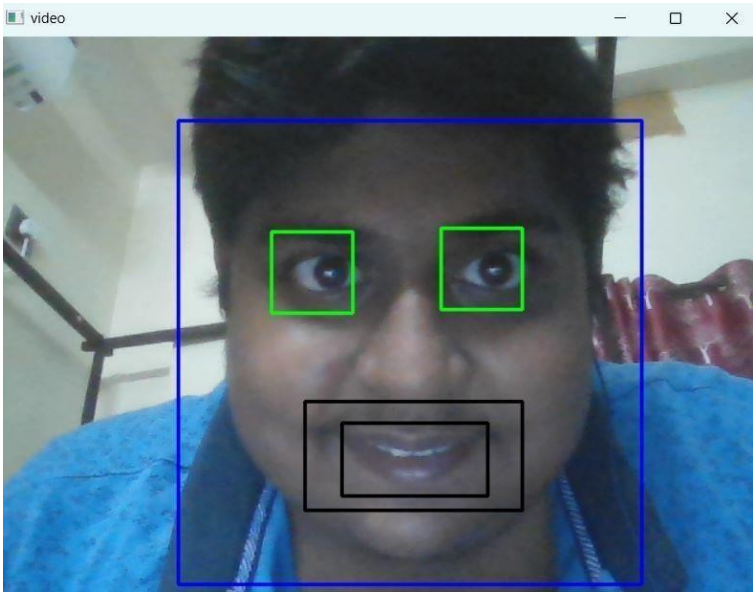
The above part will start the camera and try to detect the face ,eyes and smile together and this part

“cap.release”

“cv2.destroyAllWindows( )”

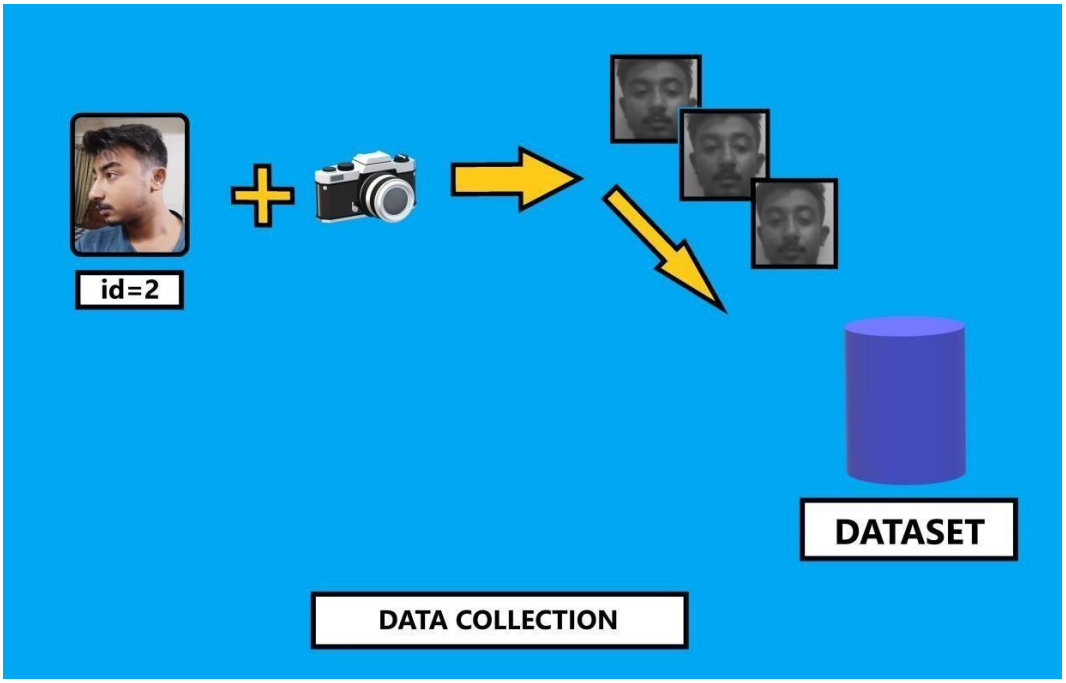
Which is using the keyboard’s ‘ESC’ button to close the running program.

The output is :



- **Dataset collection –**

Let's start our first phase of front end part. So we will do here, starting from last step (face detection) . So we are creating a dataset folder in our "FacialRecognition" folder . Where, we will simply create a dataset where we will store each id , a group of photos (taking 150 images per id) in gray with the portion that was used for face detection. Here our work is like this process :



So at first we are creating a directory at "FacialRecognition" folder named "Dataset". Then we make a python file in the "FacialRecognition" folder which is "Data_collection.py" .

So our main target here is to make a dataset of each id when the individual id's person come in front of the camera after running this "Data_collection" python file. So we are starting our code :

```
import cv2
import os

cam = cv2.VideoCapture(0)
cam.set(3, 640)
cam.set(4, 480)

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

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

print("\n [INFO] Initializing face capture. Look the camera and wait ...")

count = 0

while(True):
    ret, img = cam.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_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)
        count += 1

        cv2.imwrite("Dataset/User." + str(face_id) + '.' + str(count) + ".jpg", gray[x:x+w, y:y+h])
        cv2.imshow('image', img)

    k = cv2.waitKey(100) & 0xff
    if k == 27:
        break
    elif count >= 150:
        break

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

At first we import libraries which are OpenCV (using cv2) and os.
The code is very simple which saw for face detection. Here what we have added was an “input command” to capture the particular user id , which should be an integer number like 1,2,3 etc. So we use this part :

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

Before that we store the particular “haarcascade_frontalface_default.xml” file at the “FacialRecognition” folder and again called in the “Data_collection” python file. So we use the parameters “face_detector” and declare the haarcascade file here using “cv2.CascadeClassifier” . So we use this part as :

```
face_detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

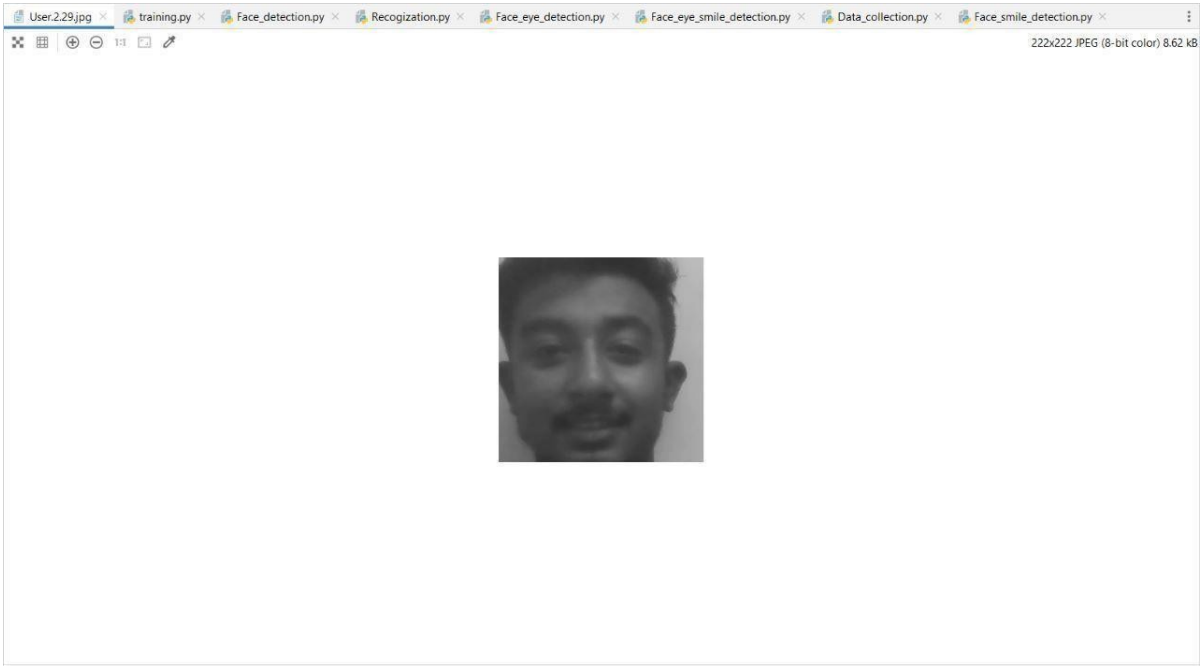
Our next aim is to store each of the captured frame in our "Dataset" folder so we are using this part :

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

Note that , for saving the above captured frame in the Dataset file we need to import that "os" library. Each file name will follow this structure "User.face_id.count.jpg" , which is ,

```
'User." + str(face_id) + '.' + str(count) + ".jpg"
```

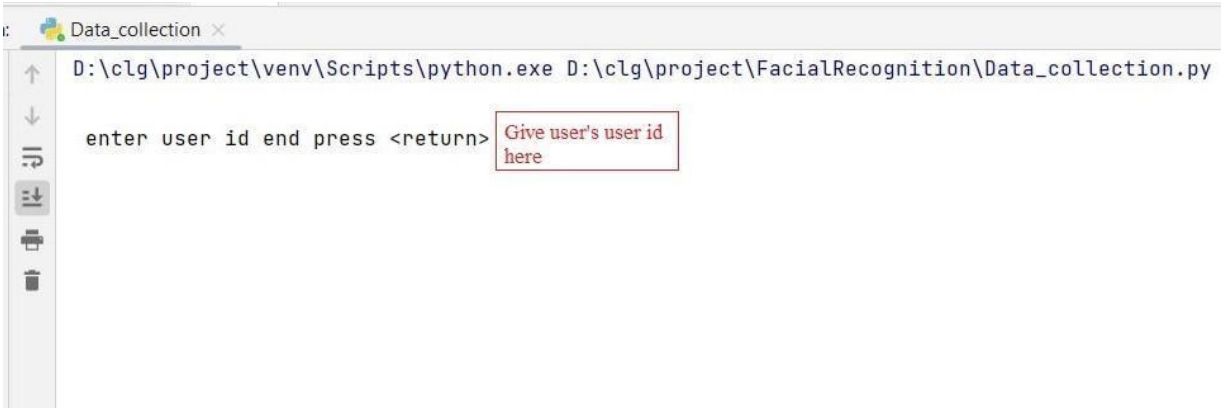
For example a user with a face_id = 2, the 29th sample file on Dataset directory will be something like this :



On our code , we are capturing 150 samples from each id . If someone wants to change the capturing image number it can do on the last "elif " means this part :

```
elif count >= 150:
    break
```

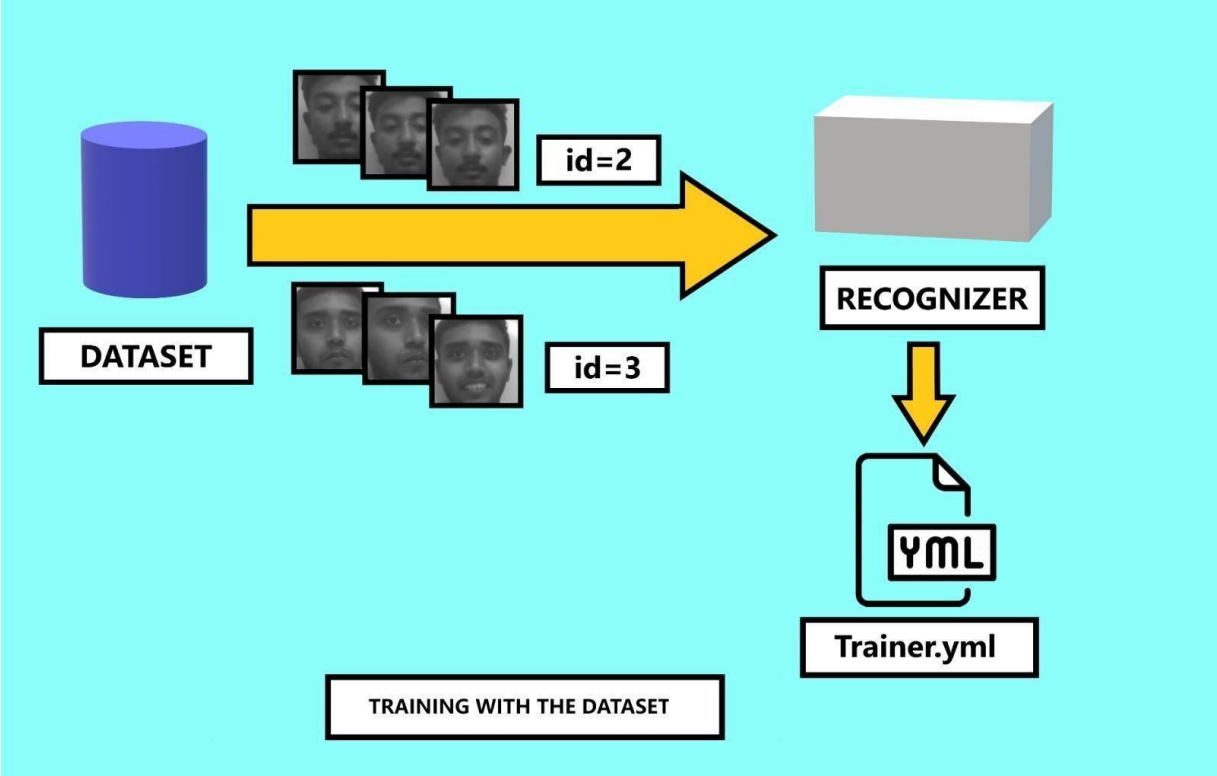
The number of samples is used to break the loop where the face samples are captured and when we need to run this python file every time when a new user comes he or she need to give his or her user id which is the next integer number of last user's user id. For example, taking that user A is the first user so when user A comes and run this python file when this part is showing :- **"enter user id end press <return>"** then user A give its id 1 and then press "Enter". Again when the user B comes then same work is done but user B needs to give its id number 2 or else. But that should be an integer value. So the output is :



And then after pressing Enter running the camera and this should give us the output as :

- **Training our dataset –**

This is the second phase of our front-end part. So we have some data of face of the users in our dataset. And now our aim is to make a training file from the dataset by recognition using "OpenCV Recognizer". So this is directly done by a specific OpenCV function which gives us ".yml" file which stores the training dataset. Here our work is like this process :



That's why we are making a sub directory named "trainer" in our "FacialRecognition" folder . In this folder the training dataset means the ".yml" file is stored .

So we make a python file named "training"

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


path = 'dataset'

recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml");


1 usage
def getImagesAndLabels(path):

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

    for imagePath in imagePaths:

        PIL_img = Image.open(imagePath).convert('L')
        img_numpy = np.array(PIL_img,'uint8')

        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)

        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))
recognizer.write('trainer/trainer.yml')


print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.unique(ids))))
```

And here at first we call some libraries which are OpenCV (using cv2) , then we call numpy and then call PIL and in the PIL we import image library , and also call the os library .

If the PIL library is not installed then open your command prompt or powershell or python IDE terminal and at that place write down the following command : “pip install pillow”.

Now at first we declare the path of dataset so we use this :

```
path = 'Dataset'
```

Now we will use as a recognizer , the LBPH (which stands for LOCAL BINARY PATTERNS HISTOGRAMS) Face Recongnizer , includes on OpenCV package . But sometimes the LBPH part gives error cause the OpenCV does not install the LBPH library that’s why we use this command in command prompt or powershell or python IDE terminal :

“pip install opencv-contrib-python”

So we do this following line for recognizer :

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

And we also use “haarcascade_frontalface_default.xml” for detector . So we use this

```
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml");
```

Now the function “getImagesAndLabels(path)” will take all photos on the directory of Dataset, and running two arrays : “Ids” and “faces”. With those arrays as input we will “train our recognizer” . So we use this :

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

Then we create a for loop and checking the images and make the images into an 8bit unsigned integer means an array which is in between 0 to 255 decimal. So we use this part :

```
    for imagePath in imagePaths:

        PIL_img = Image.open(imagePath).convert('L')
        img_numpy = np.array(PIL_img,'uint8')

        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)

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

    return faceSamples,ids
```

And also checking the Id of the images. Then the training file which named is “trainer” need to save in our trainer folder . So we use this part :

```
recognizer.write('trainer/trainer.yml')
```

After saving this model the output shows this line at the end :

“[INFO] {0} faces trained. Exiting Program”

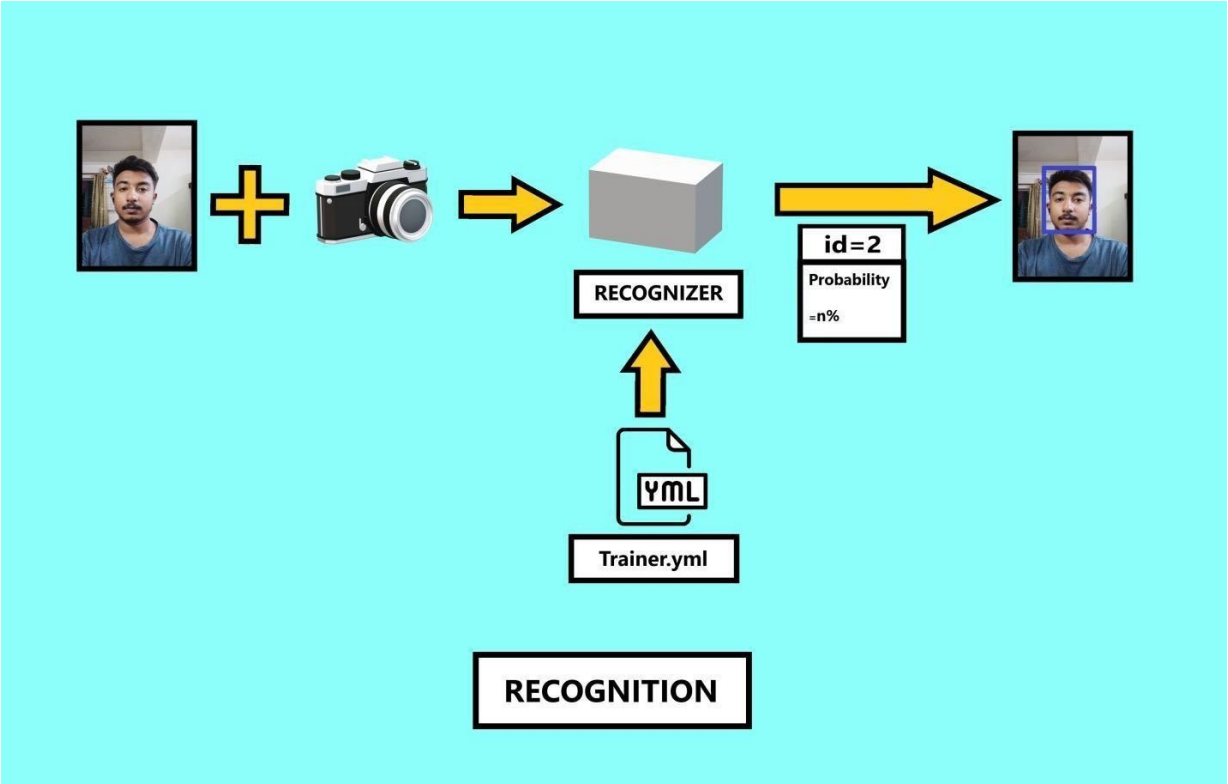
```
print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.unique(ids))))
```

Now we will train our model and save it. So we are towards the last and final phase of the fornt-end part of our project. So our work is recognising the face .

Lets start it.

- **Recognizer –**

It is the final phase of our project . Where, we will capture a fresh face on our device camera and if this person had captured face image in our dataset before the training of our model , our recognizer will make a “prediction” returning its “id” and “index” where the index shows the percentage of recognition of face in case of a match . The structure will be :



So we make a python file named “Recognition”. So we write this code :

```
import cv2
import numpy as np
import os

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);

font = cv2.FONT_HERSHEY_SIMPLEX

id = 0
names = ['None', 'Surjya', 'Subhrajyoti', 'Subradeep', 'Wasim', 'Sudipta', 'Sudip_Sir', 'Tridip_Sir', 'Suparna_mam']

cam = cv2.VideoCapture(0)
cam.set(3,640)
cam.set(4,480)

minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)

while True:
    ret, img = cam.read()
    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[x:x+w, y:y+h])

        if(confidence < 100):
            id = names[id]
            confidence = " {0}%".format(round(100 - confidence))
        else:
            id = "unknown"
            confidence = " {0}%".format(round(100 - confidence))

        cv2.putText(img, str(id), (x+5,y-5), font, 1, (255,255,255),2)
        cv2.putText(img, str(confidence), (x+5,y+h-5), font, 1, (255,255,0), 1)

    cv2.imshow('camera',img)

    k = cv2.waitKey(10) & 0xff
    if k == 27:
        break

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

Where at first we import some libraries which are OpenCV (using cv2) , then numpy and then os.

Then we again call the recognizer where we are detecting the face using LBPHFaceRecognizer (which stands for LOCAL BINARY PATTERNS HISTOGRAMS) under the OpenCV library. Then we read our training file in between the recongizer parameter so we use "recognizer.read('path of the training file')" . Then we also call the cascade path of haarcascade_frontalface_default file using this part : "cascadePath = "haarcascade_frontalface_default.xml" ". Then we again read the cascade path using OpenCV and cascade classifier. And which is declared in faceCascade parameter . So we use this part "faceCascade = CascadeClassifier(cascadePath)".

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
```

Then we are including here a new a new array so we will display the names of each id , instead of the user's id number . So we will use the following part :

```
id = 0
names = ['None', 'Surjya', 'Subhrajyoti', 'Subradeep', 'Wasim', 'Sudipta']
```

So for example here 'Surjya' will be the user with user id = 1, 'Subhrajyoti' will be the user with user id = 2, 'Subhradeep' will be the user with user id = 3 , 'Wasim' will be the user with user id = 4 , 'Sudipta' will be the user with user id = 5 .

Next , we will detect a face , which we did before with haarcascade face classifier . Having a detected face we can call the most important function.

Now , The recognizer.predict( ) , will take as a parameter , a captured portion of the face to be analysed and will return its probable owners , indicating with his or her id's name and also the predict percentage for the match . So we use this :

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

Where the recognizer.predict calls the gray portion of the face.

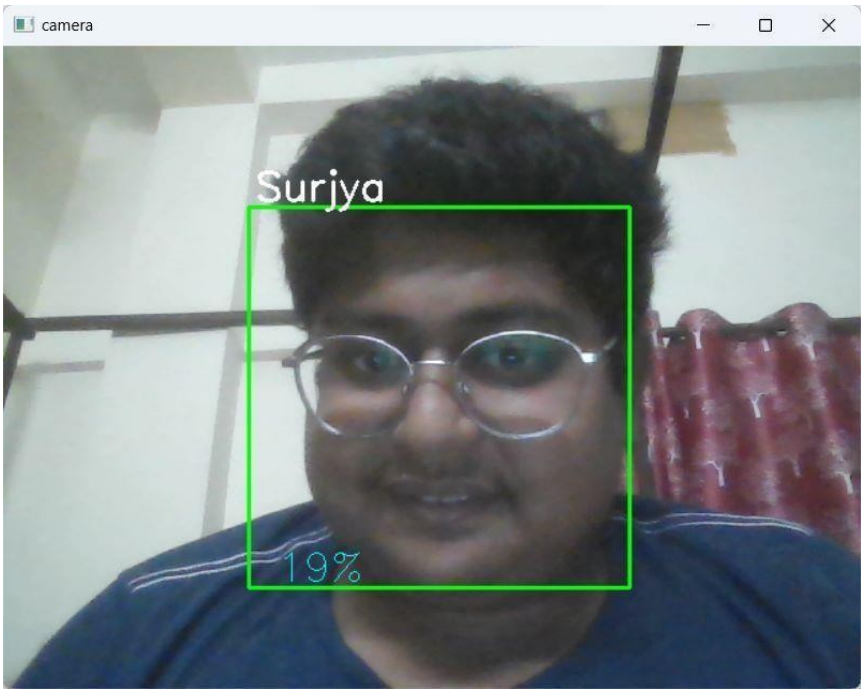
Note that the confidence index will return “zero” if it will be considered as a perfect match.

And now at last , if the recognizer could predict a face , we put a text over the image with the probable id’s name and how much is the “probability” in percentage that the match is correct. Which use this formula “confidence = 100 – confidence index” and if it is not recognize then it shows an “unknown” label above the undetected face. So the code is :

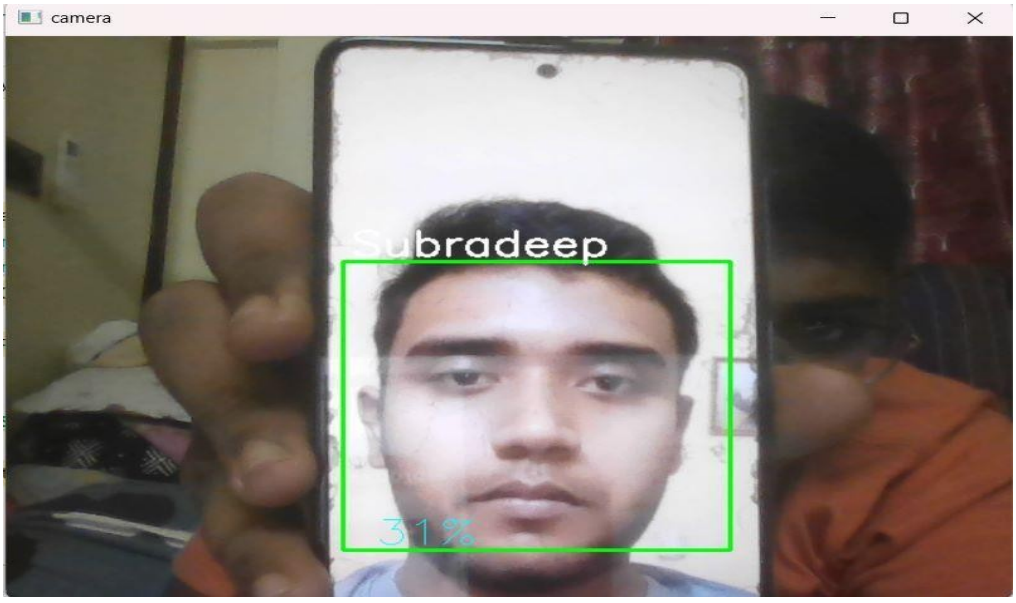
```
if(confidence < 100):
    id = names[id]
    confidence = " {0}%".format(round(100 - confidence))
else:
    id = "unknown"
    confidence = " {0}%".format(round(100 - confidence))
```

So here our third phase has come to an end. So now its time to test our project.

Now we are running the “recognition.py” and we have tested some faces and see that its working finely.



RESULT



CONCLUSION

The project was a great learning experience for us as we got to explore different aspects of computer vision and apply them to a real-world problem. W learned how to use OpenCV functions, how to implement various face detection and recognition algorithms, how to evaluate their performance, and how to optimize them for speed and accuracy.

In this project, we used OpenCV to implement face detection and recognition. We applied different methods such as Haar-Cascades and Local Binary Pattern Histograms to extract features from face images and compare them. We evaluated the accuracy and speed of each method on a dataset of faces. The project demonstrated the potential of OpenCV for face recognition applications.

FUTURE WORK

1. Enhancing Accuracy: Face recognition systems can always benefit from improvements in accuracy. You can explore advanced machine learning techniques, such as deep learning architectures, to further enhance the accuracy of your model. Experiment with different network architectures, training strategies, and data augmentation techniques to achieve better results.

2. Robustness to Variations: Make your face recognition system more robust to variations in lighting conditions, pose, expression, and occlusions. This can involve developing techniques to handle different illumination scenarios, pose estimation algorithms, and methods to handle partial occlusions or facial disguise.

3. Real-time Performance: Optimize your face recognition system to achieve real-time performance, allowing it to process video streams or live camera feeds efficiently. This could involve exploring techniques such as model compression, hardware acceleration, or parallel processing to speed up the recognition process.

4. Privacy and Security: Address the concerns related to privacy and security in face recognition systems. Consider incorporating privacy-enhancing techniques like differential privacy to protect user information. Additionally, explore methods to detect and prevent spoofing attacks, such as using liveness detection to ensure that the system is presented with a real face.

5. Demographic Bias Mitigation: Face recognition systems have been known to exhibit biases, particularly concerning race and gender. Work on identifying and mitigating these biases to ensure fair and equitable performance across different demographic groups. This could involve careful dataset curation, algorithmic adjustments, and evaluation metrics that consider fairness.

6. Multi-Modal Recognition: Explore the fusion of face recognition with other biometric modalities such as voice or gait recognition. This multi-modal approach can provide improved recognition accuracy and robustness.

7. Edge Computing: Investigate techniques for deploying face recognition models on resource-constrained devices or at the edge of the network. This allows for faster response times, increased privacy, and reduced dependence on cloud infrastructure.

8. Cross-Domain Applications: Apply face recognition techniques to various domains beyond security, such as healthcare, retail, entertainment, or education. Explore how face recognition can be utilized to improve personalized experiences, access control, or customer service.

REFERENCE

[1] About facial recognition system – https://en.wikipedia.org/wiki/Facial_recognition_system

[2] Main resource for our project - https://towardsdatascience.com/real-time-face-recognition-an-end-to-end-project-b738bb0f7348

[3] Video tutorial - https://pythonprogramming.net/loading-video-python-opencv-tutorial/

[4] MJRoBot GitHub page – https://github.com/Mjrovai/OpenCV-Face-Recognition

[5] MJRoBot projects Blog page – http://mjrobot.org/

[6] Another resource – https://www.geeksforgeeks.org/face-detection-using-python-and-opencv-with-webcam/

[7] OpenCV haarcascade GitHub link to download file- https://github.com/opencv/opencv/tree/master/data/haarcascades

[8] About Haarcascade face and eye detection : https://www.numpyninja.com/post/face-detection-using-haar-cascades-opencv-python

[9] About the NumPy – https://www.w3schools.com/python/numpy/numpy_intro.asp

https://en.wikipedia.org/wiki/NumPy

[10] About OpenCV – https://opencv.org/

[11] About the os module in the python - https://www.javatpoint.com/python-os-module#:~:text=Python%20OS%20module%20provides%20the,under%20Python's%20standard%20utility%20modules

[12] About NumPy datatype – https://pythonguides.com/python-numpy-data-types/

[13] Details about LBPH algorithm which can be downloaded in pdf format (provided by ScienceDirect) – https://doi.org/10.1016/j.procs.2022.12.108

[14] Other types of face recognition algorithms –
Eigenfaces : https://en.wikipedia.org/wiki/Eigenface
Eigenfaces (by ScienceDirect):
https://doi.org/10.1016/j.protcy.2012.02.023

Convolution Neural Network (CNN) :
https://www.tandfonline.com/doi/full/10.1080/21642583.2020.1836526