Vidyavardhini's College of Engineering & Technology Department of Computer Engineering

---

**Aim:** To perform Face detection on Video

**Objective:** Performing face recognition Generating the data for face recognition Recognizing faces preparing the training data Loading the data and recognizing faces.

**Theory:**

**Performing face recognition :**

Detecting faces is a fantastic feature of OpenCV and one that constitutes the basis for a more advanced operation: face recognition. What is face recognition? It's the ability of a program, given an image or a video feed, to identify a person. One of the ways to achieve this (and the approach adopted by OpenCV) is to "train" the program by feeding it a set of classified pictures (a facial database), and operate the recognition against those pictures. This is the process that OpenCV and its face recognition module follow to recognize faces. Another important feature of the face recognition module is that each recognition has a confidence score, which allows us to set thresholds in real-life applications to limit the amount of false reads.

Let's start from the very beginning; to operate face recognition, we need faces to recognize. You can do this in two ways: supply the images yourself or obtain freely available face databases. There are a number of face databases on the Internet:

The Yale face database (Yalefaces): http://vision.ucsd.edu/content/yalefacedatabase

To operate face recognition on these samples, you would then have to run face recognition on an image that contains the face of one of the sampled people. That may be an educational process, but I found it to be not as satisfying as providing images of my own. In fact, I probably had the same thought that many people had: I wonder if I could write a program that recognizes my face with a certain degree of confidence.

**Generating the data for face recognition:**

So let's go ahead and write a script that will generate those images for us. A few images containing different expressions are all that we need, but we have to make sure the sample images adhere to certain criteria:

- Images will be grayscale in the .pgm format
- Square shape
- All the same size images (I used 200 x 200; most freely available sets are smaller than that)

## Recognizing faces:

OpenCV 3 comes with three main methods for recognizing faces, based on three different algorithms: Eigenfaces, Fisherfaces, and Local Binary Pattern Histograms (LBPH). It is beyond the scope of this book to get into the nitty-gritty of the theoretical differences between these methods, but we can give a high-level overview of the concepts.

## Preparing the training data:

Now that we have our data, we need to load these sample pictures into our face recognition algorithms. All face recognition algorithms take two parameters in their train() method: an array of images and an array of labels. What do these labels represent? They are the IDs of a certain individual/face so that when face recognition is performed, we not only know the person was recognized but also who-among the many people available in our database-the person is. To do that, we need to create a comma-separated value (CSV) file, which will contain the path to a sample picture followed by the ID of that person.

## Loading the data and recognizing faces:

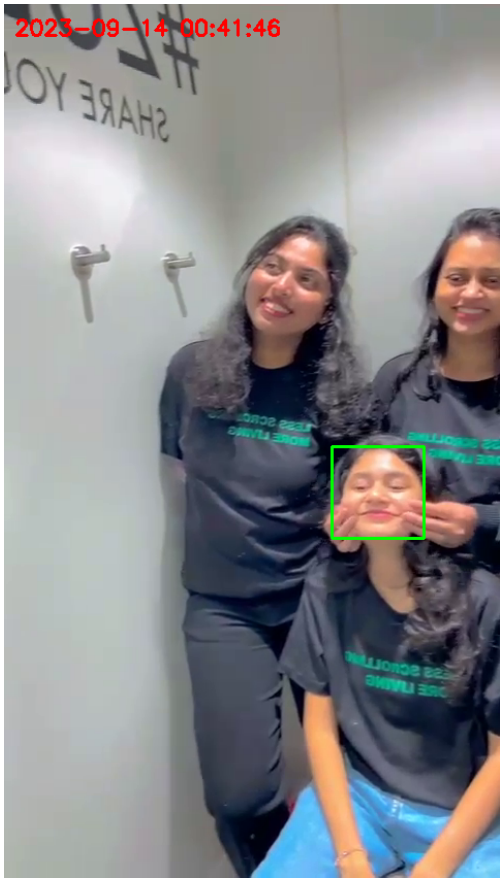
Next up, we need to load these two resources (the array of images and CSV file) into the face recognition algorithm, so it can be trained to recognize our face. To do this, we build a function that reads the CSV file and for each line of the fileloads the image at the corresponding path into the images array and the ID into the labels array.

**Code:**

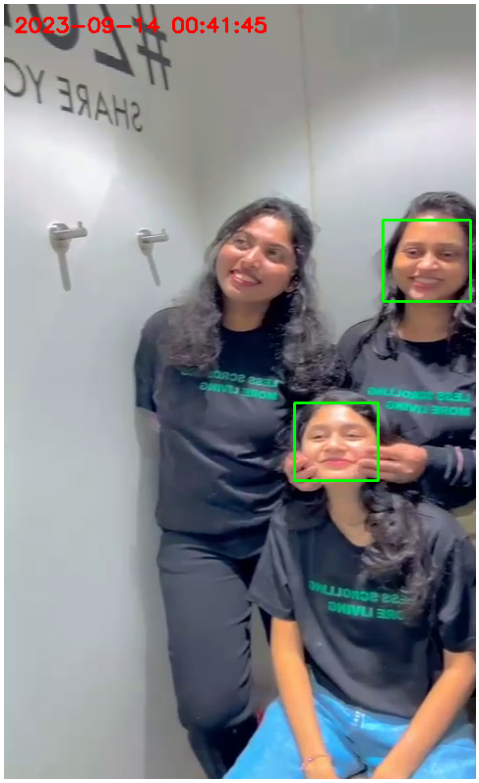
```
import cv2
import datetime
from google.colab.patches import cv2_imshow


# Load the pre-trained face detection model
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')


# Open a video capture source (use 0 for webcam or provide the path to a video file)
cap = cv2.VideoCapture('/content/_import_6174efba9bd1e6.92669478.mov') #
Replace 'your_video.mp4' with the path to your video file


while True:
    # Read a frame from the video source
    ret, frame = cap.read()


    # Break the loop if we reach the end of the video
    if not ret:
        break


    # Convert the frame to grayscale for face detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


    # Detect faces in the frame
```

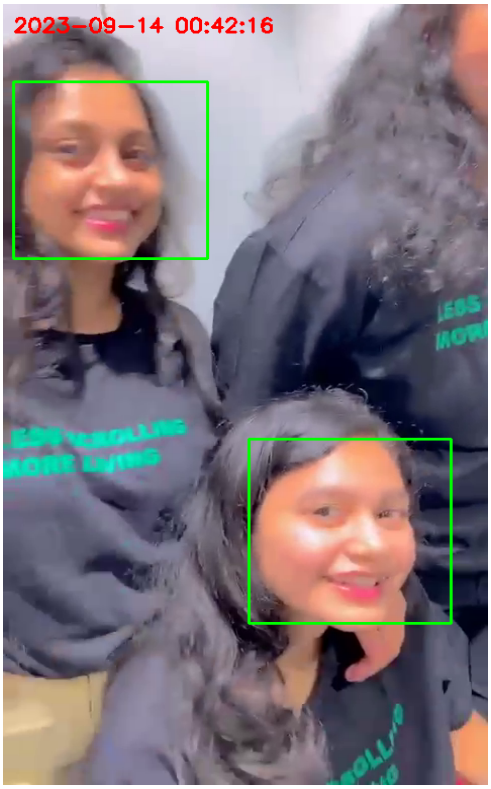
```
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5, minSize=(30, 30))

    # Add a timestamp to the frame
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    cv2.putText(frame, timestamp, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # Draw rectangles around detected faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the frame with faces and timestamp
    cv2_imshow(frame)

    # Exit the loop if the 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video source and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

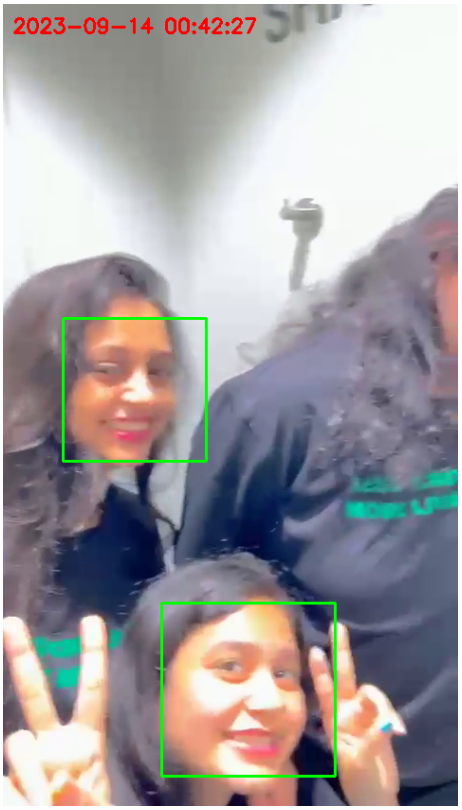
**Output:**

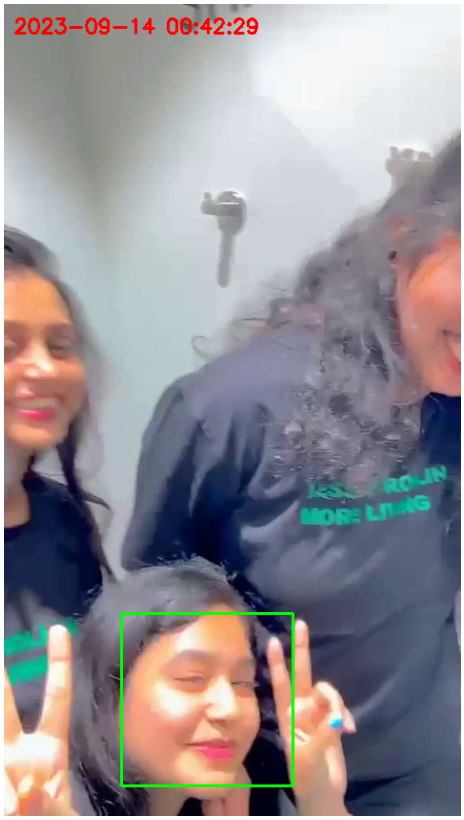

2023-09-14 00:41:46

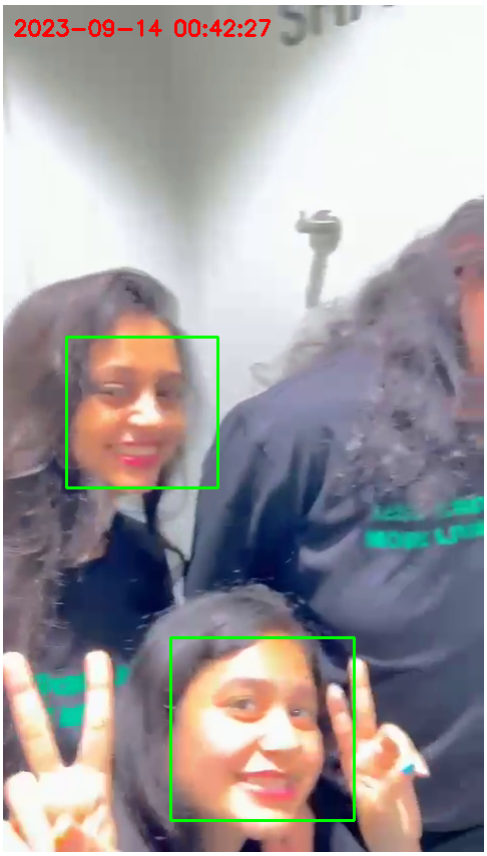

2023-09-14 00:41:50

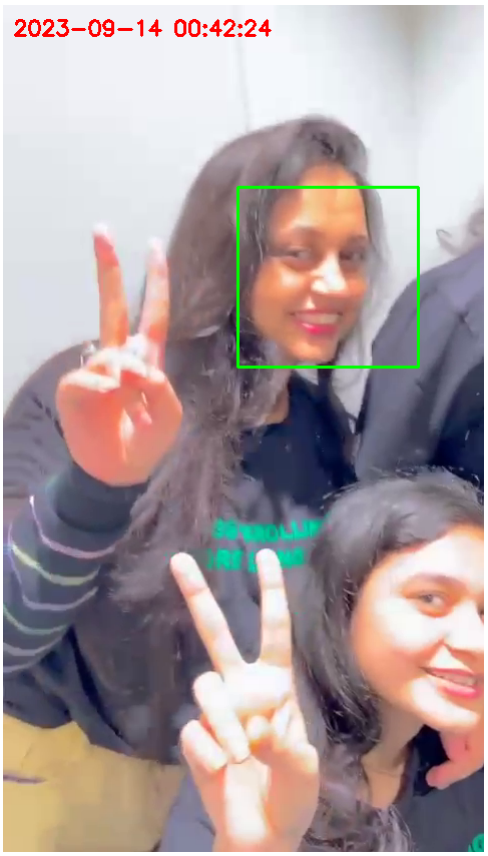

2023-09-14 00:41:45

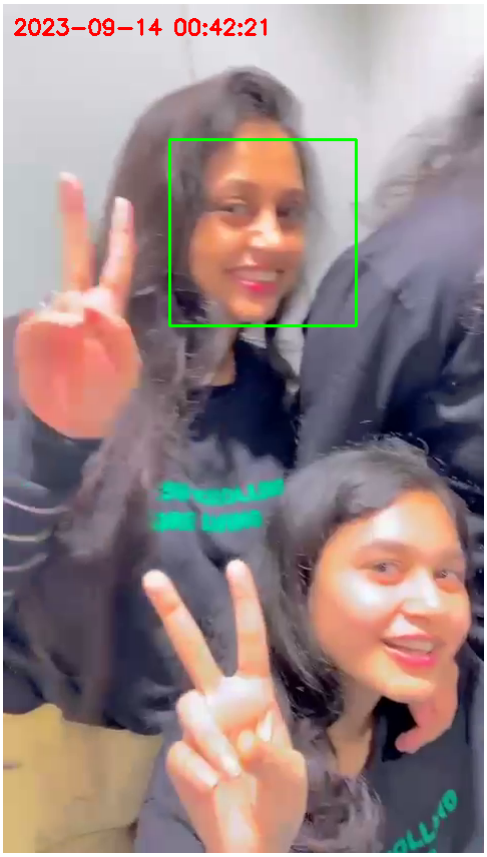

2023-09-14 00:42:16

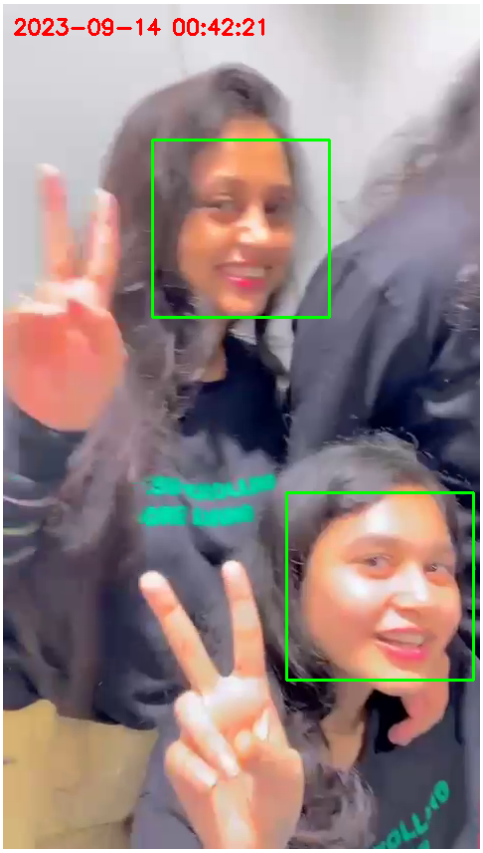

2023-09-14 00:42:27


2023-09-14 00:42:29


2023-09-14 00:42:27


2023-09-14 00:42:24




## Conclusion:

In this experiment, we have implemented real-time detection in video using OpenCV. We explored the basics of face recognition and the importance of training a program with a labeled facial database. We also discussed the generation of standardized training data and touched upon different face recognition algorithms. By creating a CSV file to map image paths to face IDs and loading the data into OpenCV, we gained valuable insights into the foundation of facial recognition technology. This experiment plays an advanced role in security, access and control of human-computer interaction.