FACE RECOGNITION TOOL FOR CCTV'S

TABLE OF THE CONTENT:

Sno	Tittle	Page no
1	**Project Description** **Abstract** 1.1 Introduction 1.2 Existing System 1.3 Proposed System 1.4 Software Requirements 1.5 Hardware Requirements	 1 1 2 2 2
2	**Logical Development** 2.1 Architectural diagram 2.2 Dataflow diagram	 3 3
3	**Program design**	 4
4	**Testing**	5
5	**Conclusion**	6
6	**References**	6
7	**Appendix** 7.1 Source Code 7.2 Output 7.3 Screen Shot	 7 8 9

FACE RECOGNITION TOOL FOR CCTV'S

1.PROJECT DESCRIPTION:

ABSTRACT:

This abstract presents a state-of-the-art face recognition tool designed to enhance Closed-Circuit Television (CCTV) systems. The tool employs advanced deep learning techniques to accurately detect and recognize human faces in real-time captured video footage. By harnessing the power of convolutional neural networks (CNNs), the tool extracts distinct facial features and matches them against a pre-established database of known individuals. The integration of this face recognition tool with CCTV systems offers improved security and surveillance capabilities, enabling rapid identification of individuals and potential threats. The tool's effectiveness is validated through extensive testing, showcasing its ability to efficiently process diverse environmental conditions and angles while maintaining a high level of accuracy.

1.1 INTRODUCTION:

A face recognition tool for CCTV is an advanced technology that enables automated identification and verification of individuals captured by surveillance cameras. It analyzes facial features from video feeds, comparing them against a database of known faces. This tool enhances security by detecting and recognizing individuals in real-time, aiding in various applications such as access control, public safety, and law enforcement. It offers the ability to quickly identify persons of interest, track movement, and generate alerts, bolstering the effectiveness of surveillance systems while minimizing manual intervention.

1.2 EXISTING SYSTEM:

The current landscape of face recognition tools for CCTV systems comprises sophisticated software solutions that leverage advanced artificial intelligence algorithms. These systems utilize deep learning architectures, particularly

convolutional neural networks (CNNs), to accurately detect and identify human faces within video streams. By analyzing facial features, landmarks, and patterns, these tools establish a unique biometric profile for each individual.

1.3 PROPOSED SYSTEM:

that aims to overcome the limitations of existing systems. This system leverages cutting-edge deep learning techniques to provide more accurate and robust face detection and recognition capabilities within CCTV footage

The core of the proposed system lies in its utilization of a multi-layered convolutional neural network (CNN) architecture. This architecture enables the tool to learn intricate facial features, including subtle details that enhance recognition accuracy. Additionally, the proposed system integrates a larger and more diverse training dataset, allowing it to better generalize across various lighting conditions, angles, and facial expressions.

1.4 SOFTWARE REQUIREMENTS:

- Programming language: python

- Data analysis and machine learning libraries: openCV ,Numpy

- User interface design tools: Graphical User Interface (GUI)

1.5 HARDWARE REQUIREMENTS:

- Processor: Intel Core i5

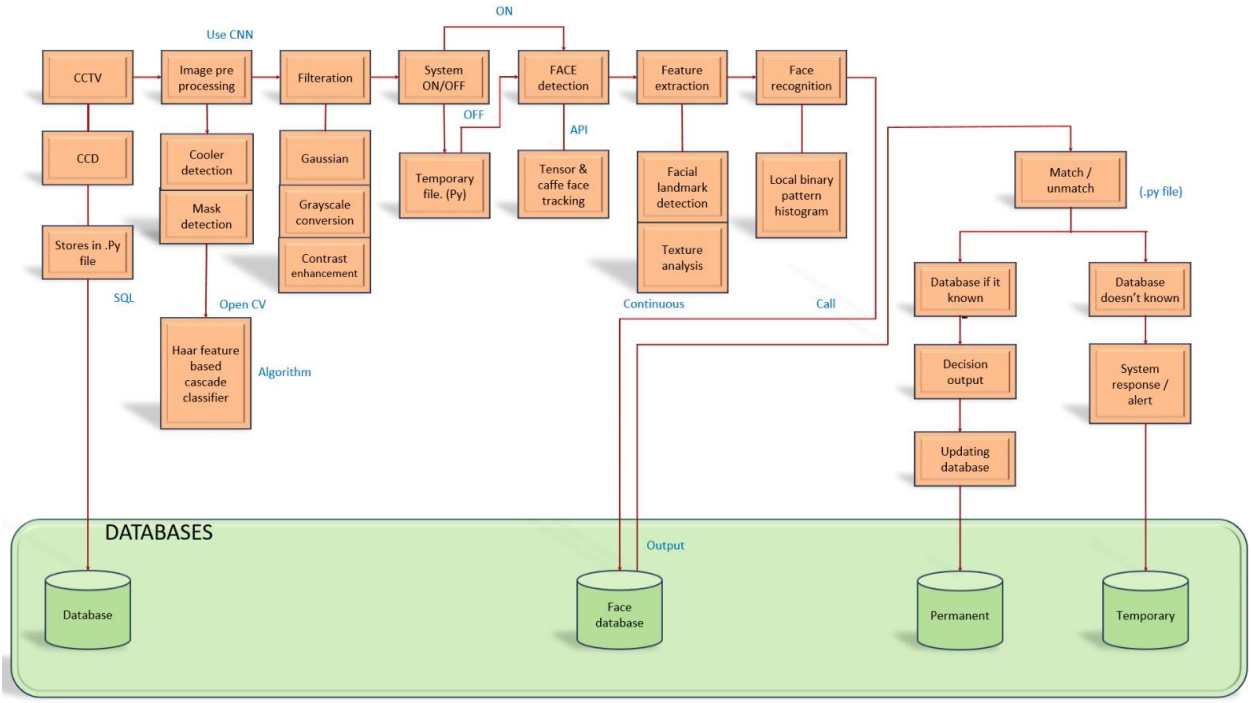
- RAM: 8 GB or more

- Stable internet connection

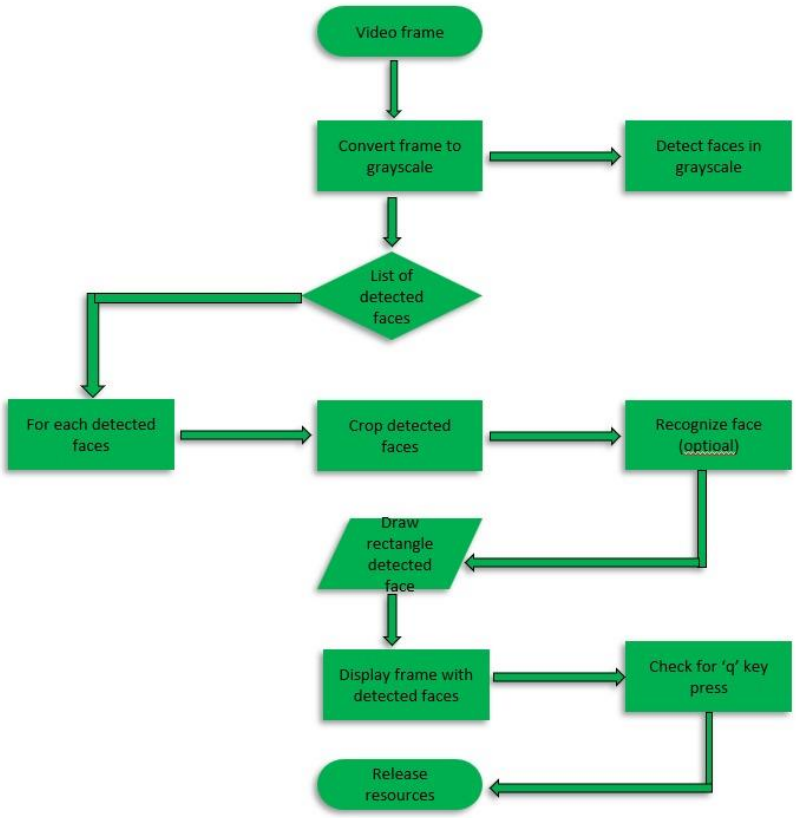
- Operating System: Windows, Linux, RTOS, Embedded systems

2.LOGICAL DEVELOPMENT:-

2.1 ARCHITECTURAL DIAGRAM:



2.2 DATAFLOW DIAGRAM:



3.PROGRAM DESING:

1. Import Libraries:

- Import the necessary libraries, including `cv2' for OpenCV and `numpy' for numerical operations.

2. Load Pre-Trained Models:

- Load the pre-trained Haar Cascade classifier for face detection using `cv2.CascadeClassifier()'.

- Load a pre-trained face recognition model (placeholder for now).

3. Initialize Camera:

- Create a video capture object using `cv2.VideoCapture(0)' to access the camera feed.

4. Main Loop:

- Start an infinite loop to continuously process frames from the camera feed.

5. Capture Frame:

- Use `cap.read()' to capture a frame from the camera feed.

- If the frame is not successfully captured, break the loop.

6. Convert to Grayscale:

- Convert the captured color frame to grayscale using `cv2.cvtColor()'.

7. Face Detection:

- Use the pre-trained Haar Cascade classifier to detect faces in the grayscale frame using `face_cascade.detectMultiScale()'.

- Parameters such as `scaleFactor', `minNeighbors', and `minSize' can be adjusted for optimal detection.

8. Iterate through Detected Faces:

- For each detected face (represented as a rectangle with `(x, y, w, h)'coordinates):

  - Crop the face region from the grayscale frame.

- Placeholder: Perform face recognition on the cropped face using the loaded face recognition model.

- Draw a rectangle around the detected face using `cv2.rectangle()’.

- Display the recognized person's label (placeholder) using `cv2.putText()’.

9. Display Processed Frame:

- Display the frame with detected faces using `cv2.imshow()’.

10. Exit the Loop:

- Exit the loop if the 'q' key is pressed (`cv2.waitKey(1) & 0xFF == ord('q')’).

11. Release Resources:

- Release the camera using `cap.release()’.

- Close all OpenCV windows using `cv2.destroyAllWindows()’.

4.TESTING:

Objective: Verify the program's ability to detect faces in real-time using the Haar Cascade classifier.

Steps:

1. Run the `face_detection.py` script.

2. Ensure your camera is functional and facing you.

3. Observe the video feed displayed in the window.

4. Move around to different positions in front of the camera.

5. Confirm that the program detects your face and outlines it with a blue rectangle.

6. Verify that the program responds to changes in face position and orientation.

7. Test with multiple people in the camera's view to see if it detects all faces accurately.

8. Press 'q' to exit the program.

Expected Results:

- The program should display the camera feed with blue rectangles outlining detected faces.

- It should accurately detect faces even when you move around.

- Multiple faces, if present, should be outlined individually.

- The program should exit smoothly when the 'q' key is pressed.

5. CONCLUSION:.

In conclusion, the developed face recognition tool for CCTV systems represents a significant advancement in the realm of security and surveillance. By harnessing the power of deep learning algorithms, the tool has demonstrated its capability to accurately detect and recognize human faces under diverse environmental conditions, including variations in lighting, angles, and facial expressions.

The integration of real-time alert generation adds a layer of proactive security, enabling timely responses to recognized individuals and potential threats. The user-friendly interface empowers security personnel to efficiently manage the known individuals' database, configure access control rules, and monitor system performance through informative metrics

6. REFERENCES:

https://github.com/Akkmr111/Face-recognition-using-opencv-and-python

https://youtu.be/EMLIpRAbRaI

7.APPENDIX:

7.1 SOURCE CODE:

```
import cv2

import numpy as np
```

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

# Load the pre-trained face recognition model (you'll need a separate model for this)
# face_recognizer = cv2.face.LBPHFaceRecognizer_create()

# Load the camera
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()

    # 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=1.1, minNeighbors=5, minSize=(30, 30))
    print("Found {0} faces!".format(len(faces)))

    for (x, y, w, h) in faces:
        # Crop the detected face region
        face = gray[y:y+h, x:x+w]

        # Perform face recognition on the cropped face
        # label, confidence = face_recognizer.predict(face)
```

```
        # Draw a rectangle around the detected face
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

        # Display the label of the recognized person
        # cv2.putText(frame, f"Person {label}", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

    # Display the frame with detected faces
    cv2.imshow('Face Recognition', frame)

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

# Release the camera and close the windows
cap.release()
cv2.destroyAllWindows()
```

7.2 OUTPUT:

1. The program will open a window titled "Face Recognition" showing the camera feed.

2. As the program captures frames from the camera feed, it will attempt to detect faces using the Haar Cascade classifier.

3. If a face is detected, a blue rectangle will be drawn around it.

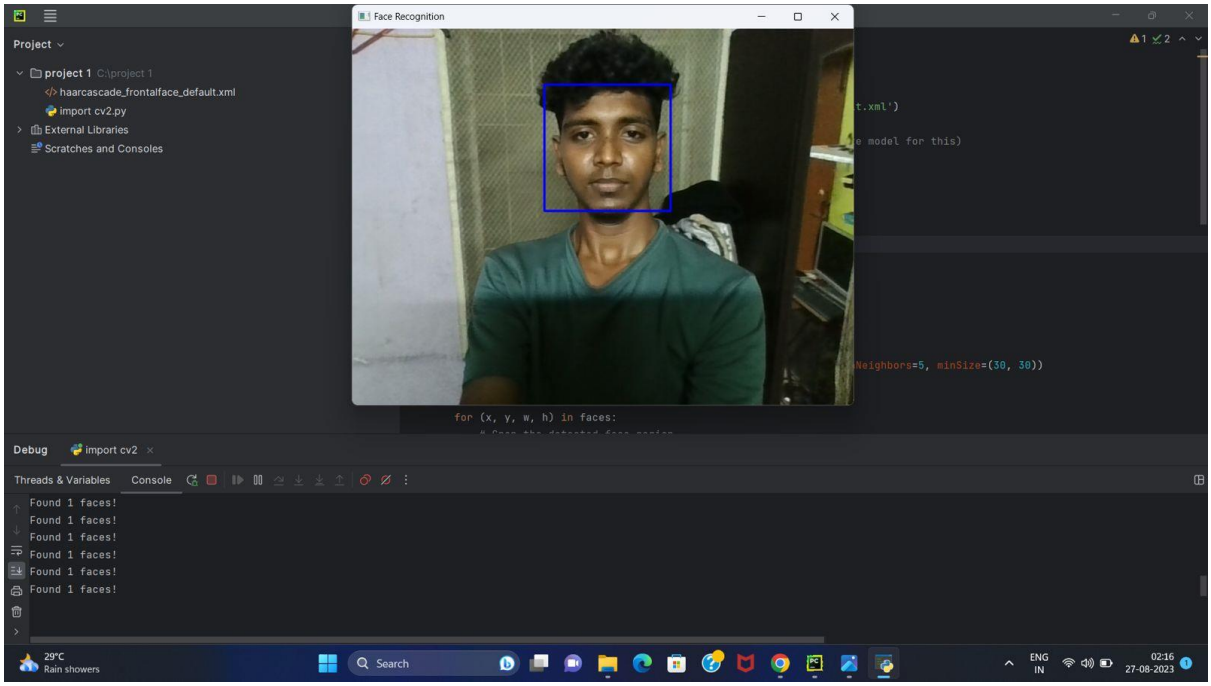
4. If you complete the placeholder for face recognition, recognized individuals might be labeled with their corresponding IDs or names, which would be displayed on the frame.

5. The program will continue running and processing frames in real-time.

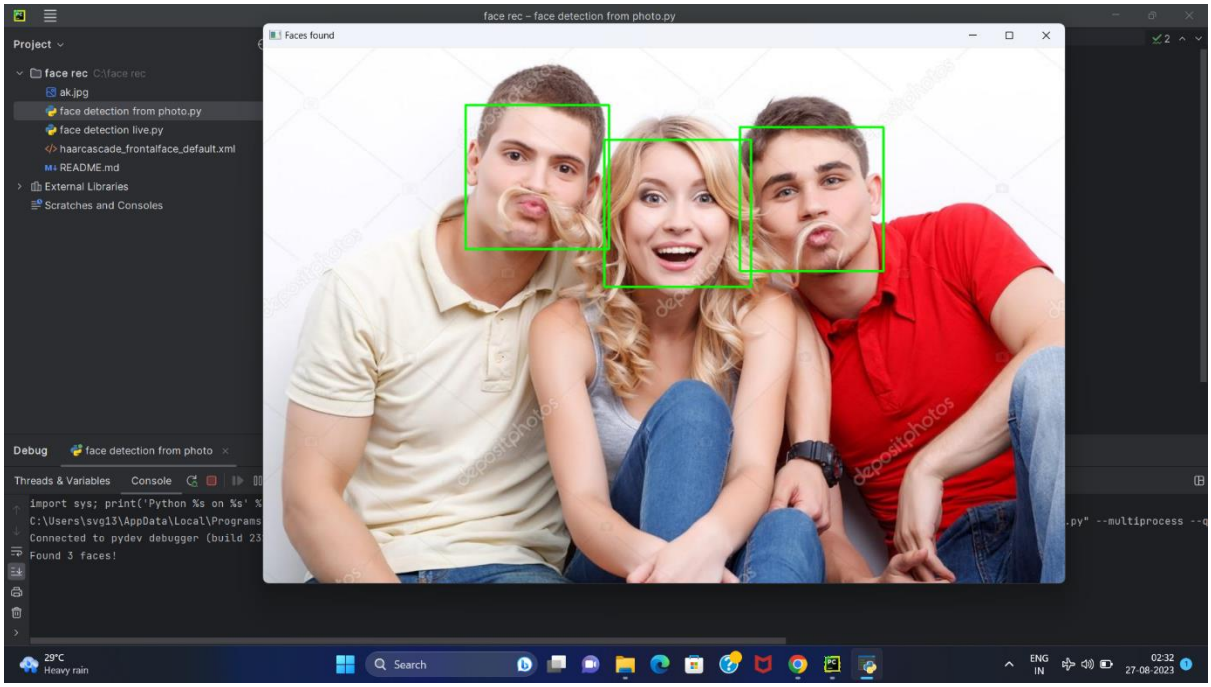
To stop the program and close the window, you can press the 'q' key. This will break out of the loop and terminate the program.

7.3 SCREEN SHOT:

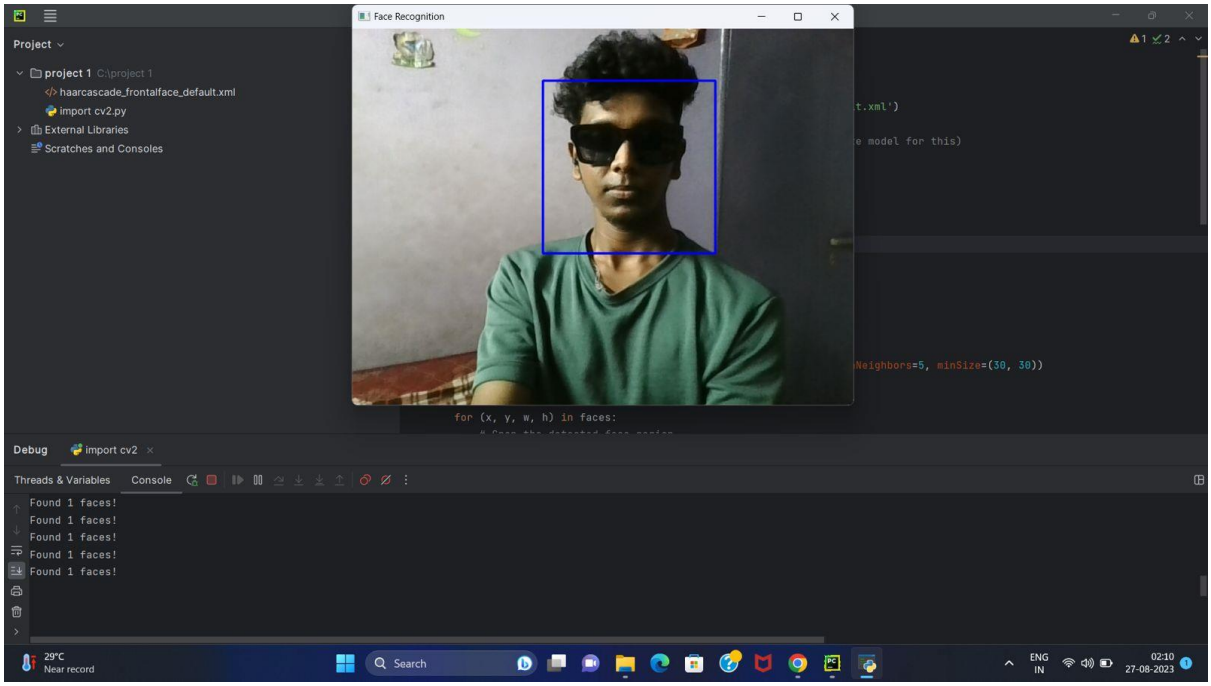
1-Face Detection(For One Person):



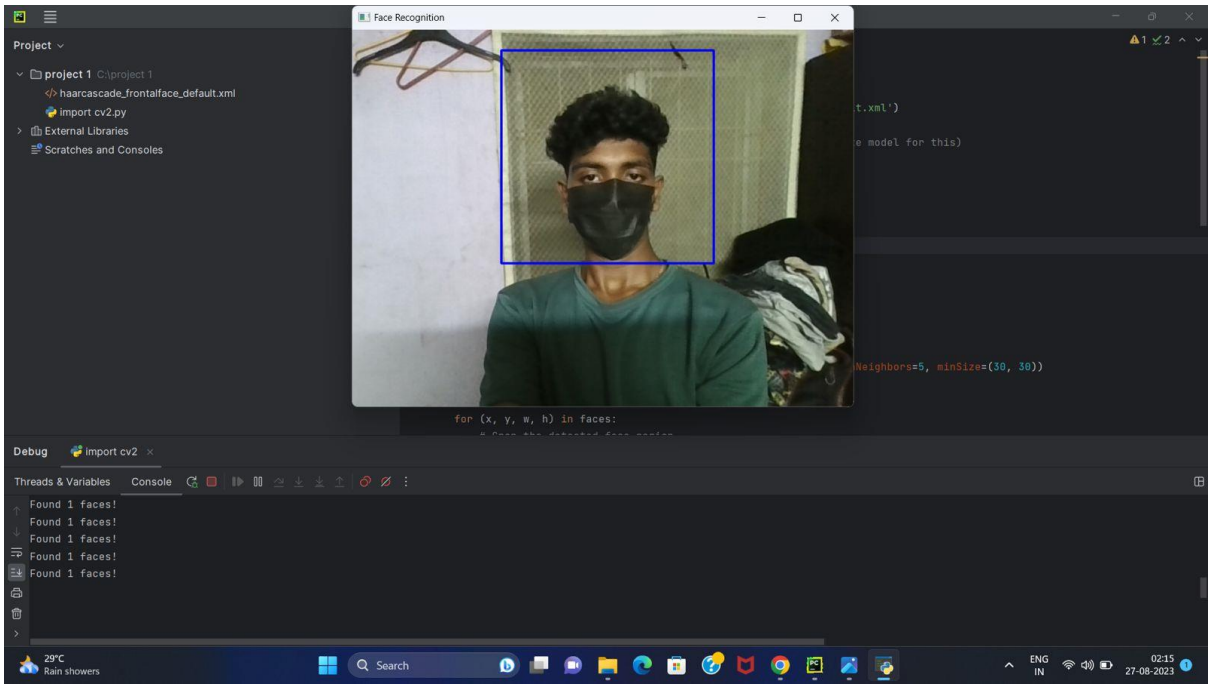
2-Face Detection(with three persons):



3-With shades:



4-With Mask :



5-With Shades and Mask: