Embedded C Project
Digital Assignment

Smart Face Recognition Security System

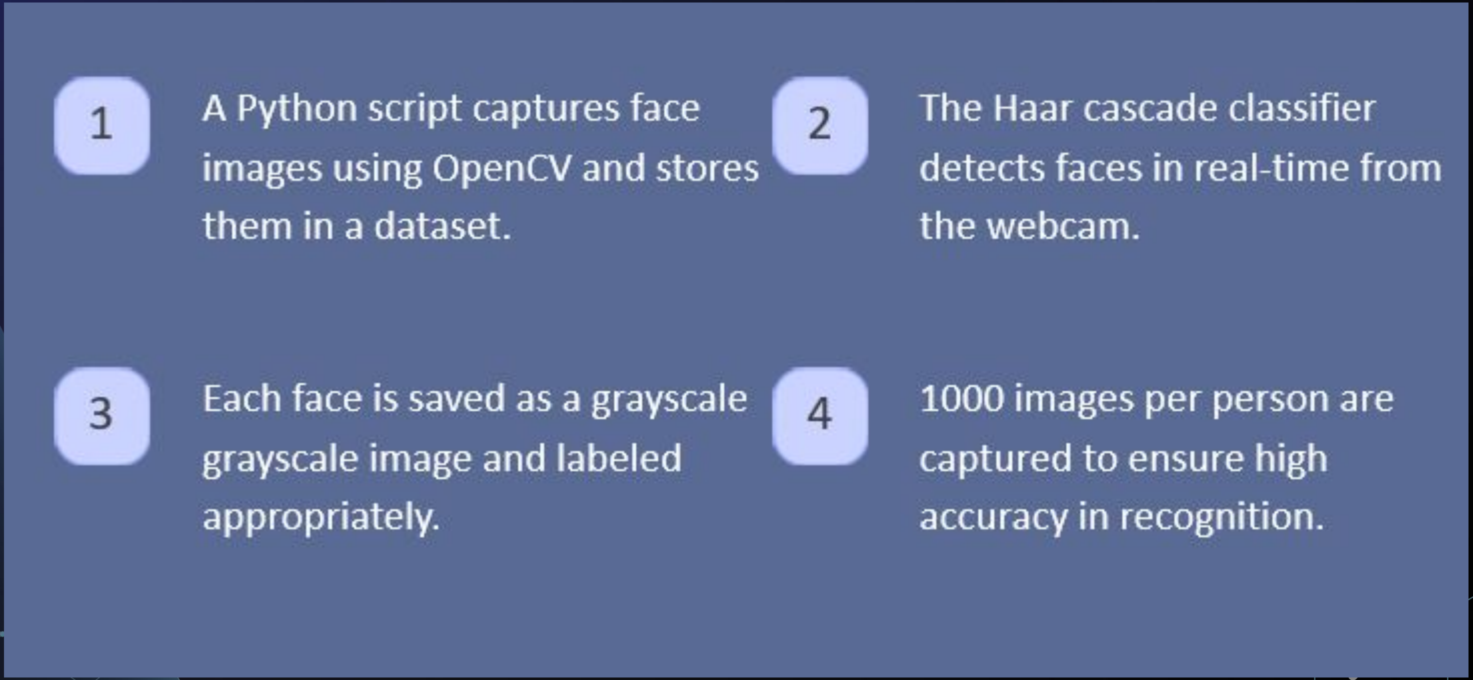
Integrating ESP32 for Real-Time Monitoring and Alerts

Kaushik R Haran , 22BML0022
Uday Sooraj , 22BEC0461
Asish Joy , 22BEC0341

Introduction

This presentation outlines the design and implementation of a face recognition-based security system that leverages the capabilities of the ESP32 microcontroller. The system is engineered to enhance security by identifying and verifying individuals through facial recognition, thereby reducing the risk of unauthorized access. The ESP32, with its built-in Wi-Fi and processing power, serves as the central controller, enabling seamless communication between hardware components and the web interface. A camera module captures facial images, which are then processed and compared against a database of authorized users. If a match is found, access is granted; otherwise, an alert is generated. The integrated web interface allows users to monitor the system in real time, view logs, and receive alerts remotely, making it a convenient and effective solution for modern security needs in both residential and commercial environments.

01 Data Collection



1 A Python script captures face images using OpenCV and stores them in a dataset.

2 The Haar cascade classifier detects faces in real-time from the webcam.

3 Each face is saved as a grayscale grayscale image and labeled appropriately.

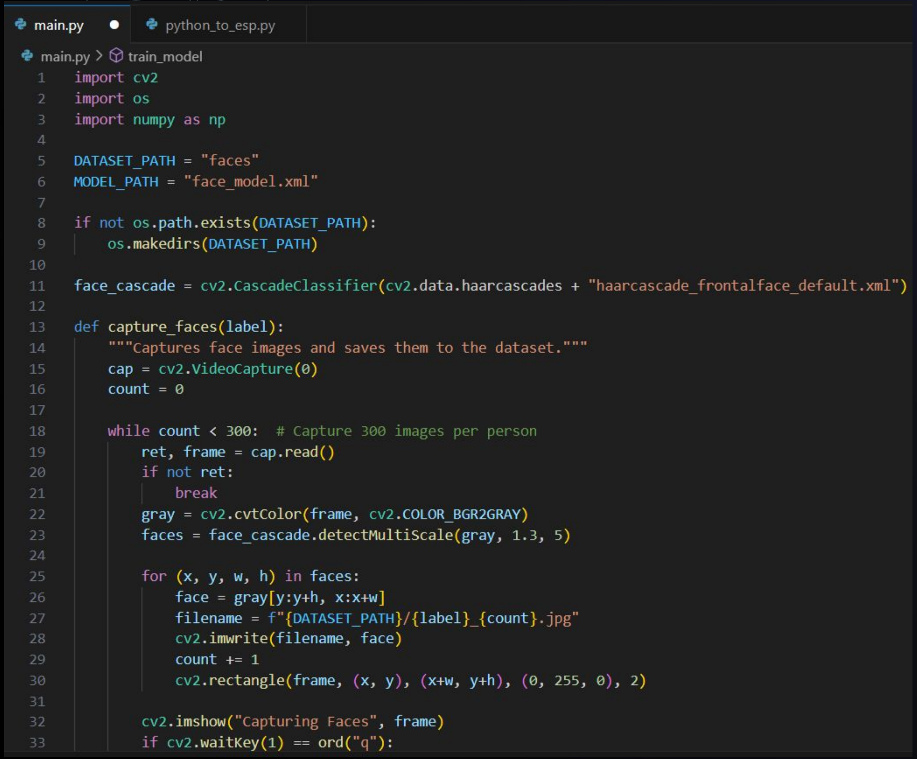
4 1000 images per person are captured to ensure high accuracy in recognition.

Face image capturing using OpenCV

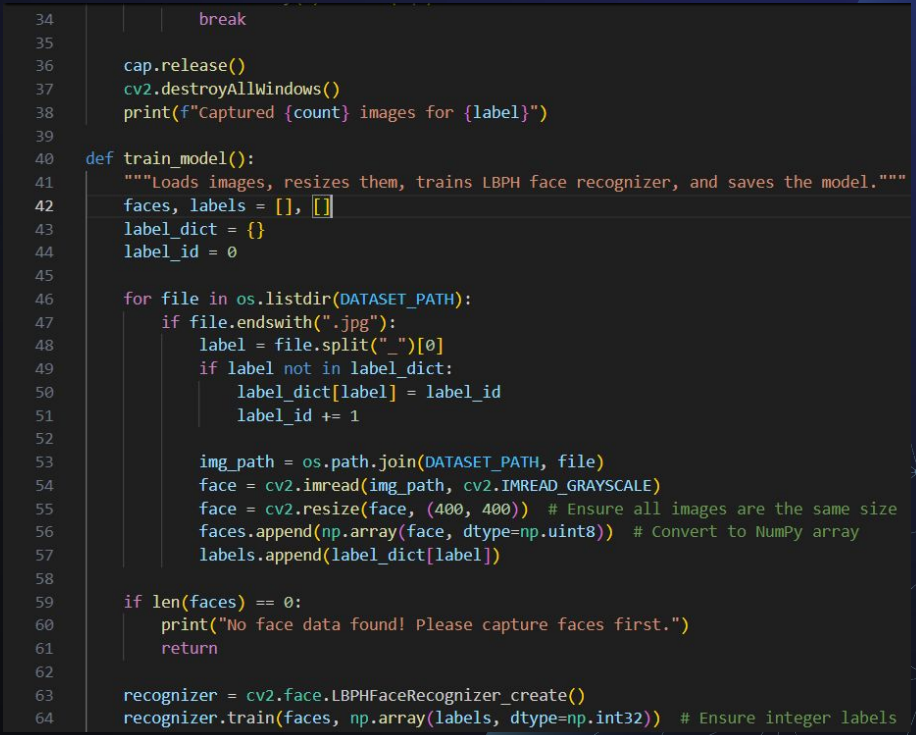
- ❑ A Python script utilizes OpenCV to access the webcam and detect faces in real time.
- ❑ Multiple images are captured per person to ensure a diverse dataset.
- ❑ Images are taken under different lighting, angles, and expressions for robustness.
- ❑ Each captured image is converted to grayscale to reduce complexity and improve processing speed.
- ❑ Images are labeled with unique IDs for each individual.

The resulting dataset is used for training the face recognition model effectively

Python Code

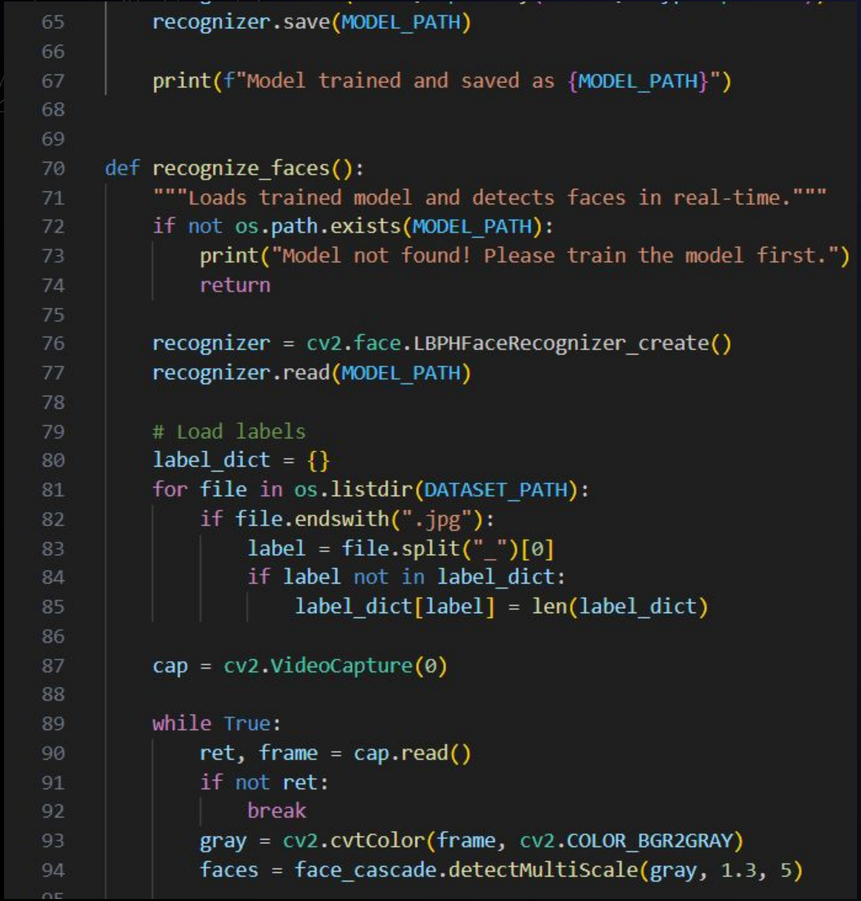


```python
import cv2
import os
import numpy as np

DATASET_PATH = "faces"
MODEL_PATH = "face_model.xml"

if not os.path.exists(DATASET_PATH):
    os.makedirs(DATASET_PATH)

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

def capture_faces(label):
    """Captures face images and saves them to the dataset."""
    cap = cv2.VideoCapture(0)
    count = 0

    while count < 300:  # Capture 300 images per person
        ret, frame = cap.read()
        if not ret:
            break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        for (x, y, w, h) in faces:
            face = gray[y:y+h, x:x+w]
            filename = f"{DATASET_PATH}/{label}_{count}.jpg"
            cv2.imwrite(filename, face)
            count += 1
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        cv2.imshow("Capturing Faces", frame)
        if cv2.waitKey(1) == ord("q"):
```

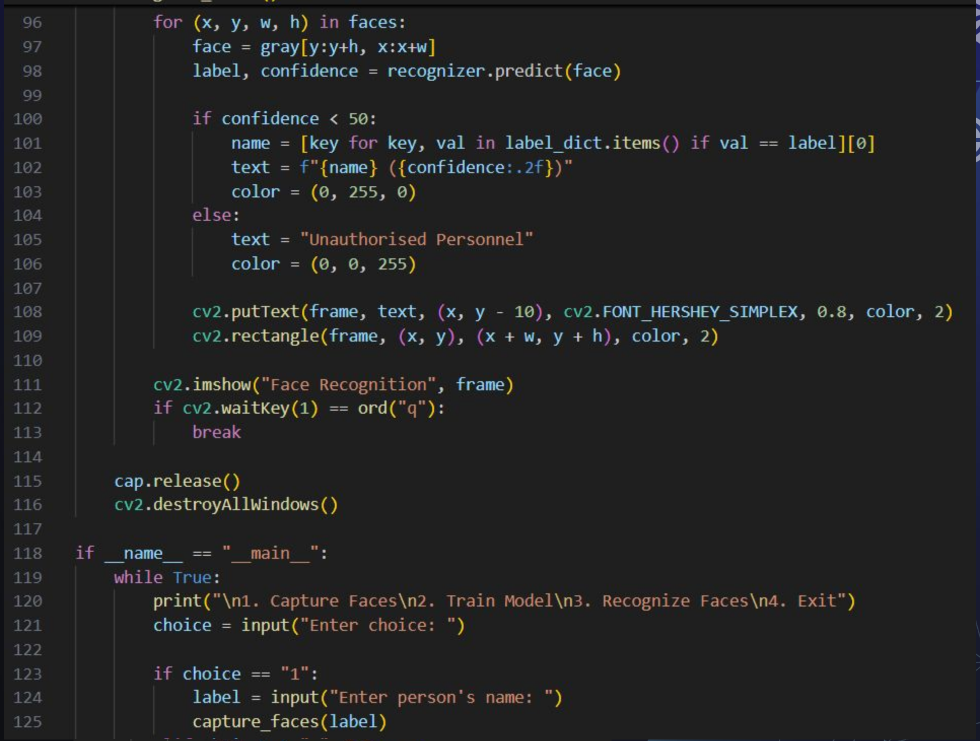


```python
            break

    cap.release()
    cv2.destroyAllWindows()
    print(f"Captured {count} images for {label}")

def train_model():
    """Loads images, resizes them, trains LBPH face recognizer, and saves the model."""
    faces, labels = [], []
    label_dict = {}
    label_id = 0

    for file in os.listdir(DATASET_PATH):
        if file.endswith(".jpg"):
            label = file.split("_")[0]
            if label not in label_dict:
                label_dict[label] = label_id
                label_id += 1

            img_path = os.path.join(DATASET_PATH, file)
            face = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            face = cv2.resize(face, (400, 400))  # Ensure all images are the same size
            faces.append(np.array(face, dtype=np.uint8))  # Convert to NumPy array
            labels.append(label_dict[label])

    if len(faces) == 0:
        print("No face data found! Please capture faces first.")
        return

    recognizer = cv2.face.LBPHFaceRecognizer_create()
    recognizer.train(faces, np.array(labels, dtype=np.int32))  # Ensure integer labels
```

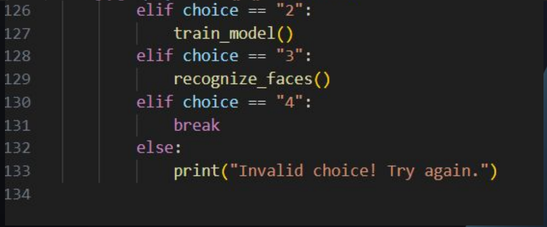
Python Code



```
    recognizer.save(MODEL_PATH)

    print(f"Model trained and saved as {MODEL_PATH}")


def recognize_faces():
    """Loads trained model and detects faces in real-time."""
    if not os.path.exists(MODEL_PATH):
        print("Model not found! Please train the model first.")
        return

    recognizer = cv2.face.LBPHFaceRecognizer_create()
    recognizer.read(MODEL_PATH)

    # Load labels
    label_dict = {}
    for file in os.listdir(DATASET_PATH):
        if file.endswith(".jpg"):
            label = file.split("_")[0]
            if label not in label_dict:
                label_dict[label] = len(label_dict)

    cap = cv2.VideoCapture(0)

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```



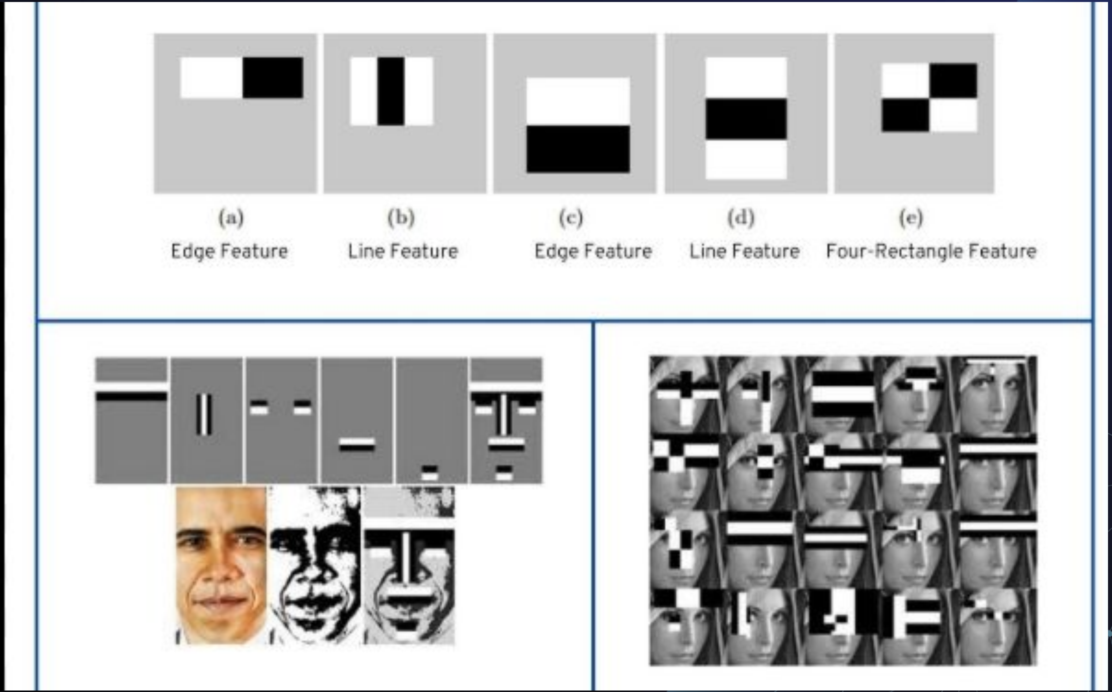
```
        for (x, y, w, h) in faces:
            face = gray[y:y+h, x:x+w]
            label, confidence = recognizer.predict(face)

            if confidence < 50:
                name = [key for key, val in label_dict.items() if val == label][0]
                text = f"{name} ({confidence:.2f})"
                color = (0, 255, 0)
            else:
                text = "Unauthorised Personnel"
                color = (0, 0, 255)

            cv2.putText(frame, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)
            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

        cv2.imshow("Face Recognition", frame)
        if cv2.waitKey(1) == ord("q"):
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    while True:
        print("\n1. Capture Faces\n2. Train Model\n3. Recognize Faces\n4. Exit")
        choice = input("Enter choice: ")

        if choice == "1":
            label = input("Enter person's name: ")
            capture_faces(label)
```



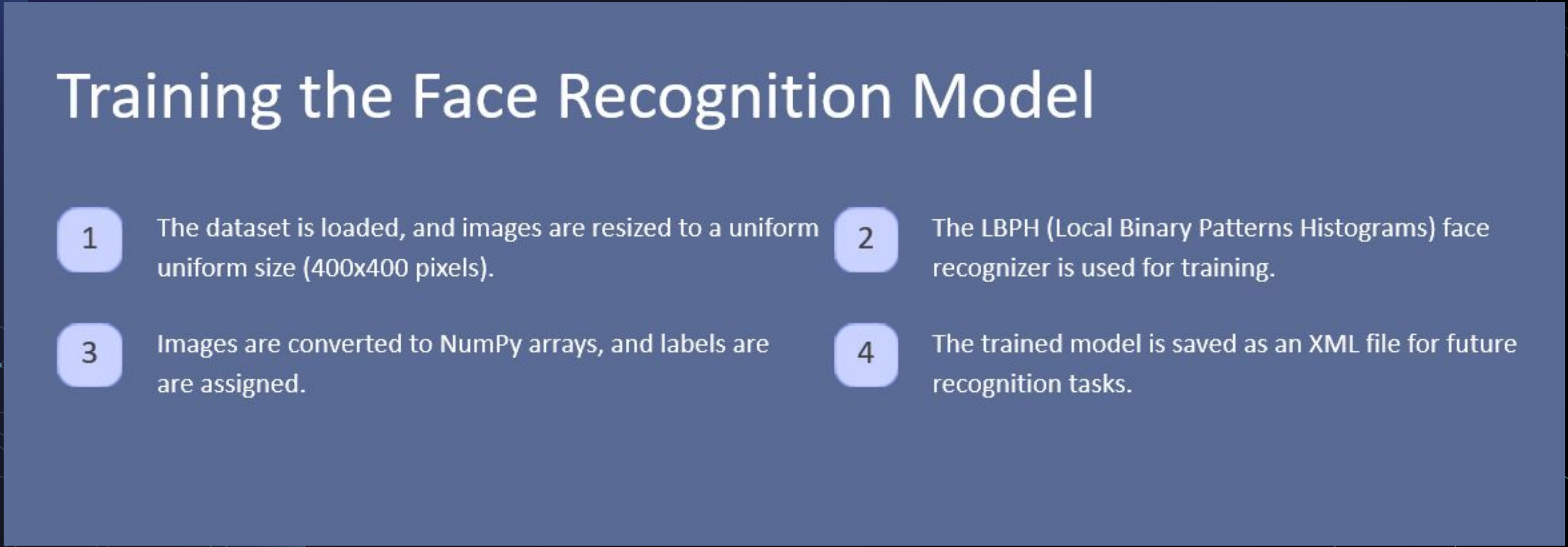
```
        elif choice == "2":
            train_model()
        elif choice == "3":
            recognize_faces()
        elif choice == "4":
            break
        else:
            print("Invalid choice! Try again.")
```

Real-time face detection with Haar cascade classifier

The Haar cascade classifier is employed to detect faces in real-time from the webcam feed. This method is based on machine learning and utilizes pre-trained models to efficiently recognize facial features. By analyzing frames from the video feed, the system can quickly identify and differentiate faces. To enhance accuracy, the system captures around 1000 images per person, ensuring a robust dataset for precise recognition. This approach improves the reliability of face detection, reducing false positives and enhancing the overall performance of the recognition system..



02 Face Recognition



Training the Face Recognition Model

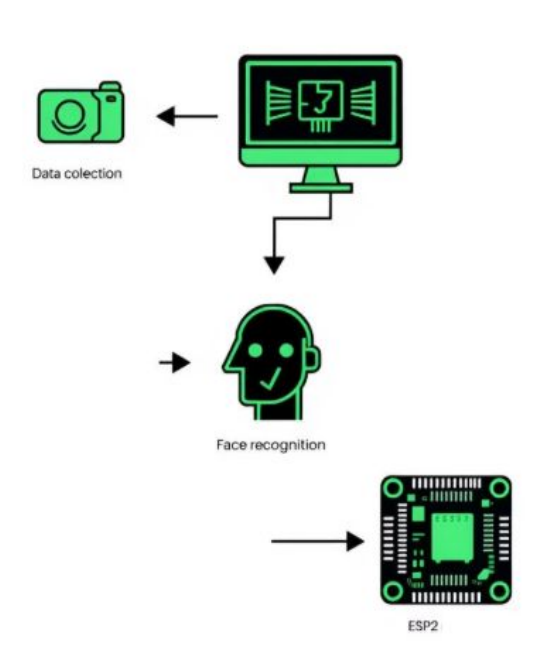
1. The dataset is loaded, and images are resized to a uniform uniform size (400x400 pixels).

2. The LBPH (Local Binary Patterns Histograms) face recognizer is used for training.

3. Images are converted to NumPy arrays, and labels are are assigned.

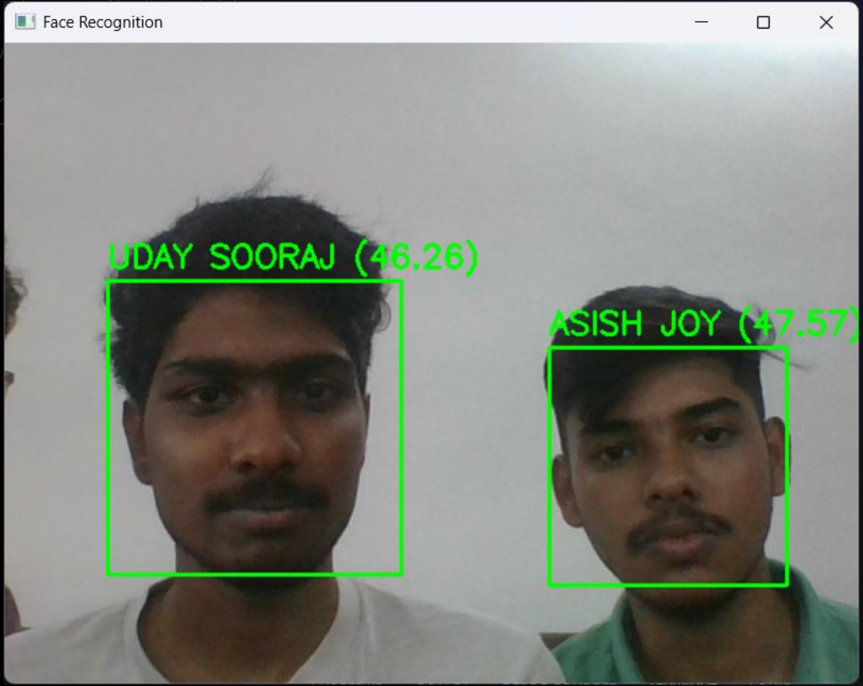
4. The trained model is saved as an XML file for future recognition tasks.

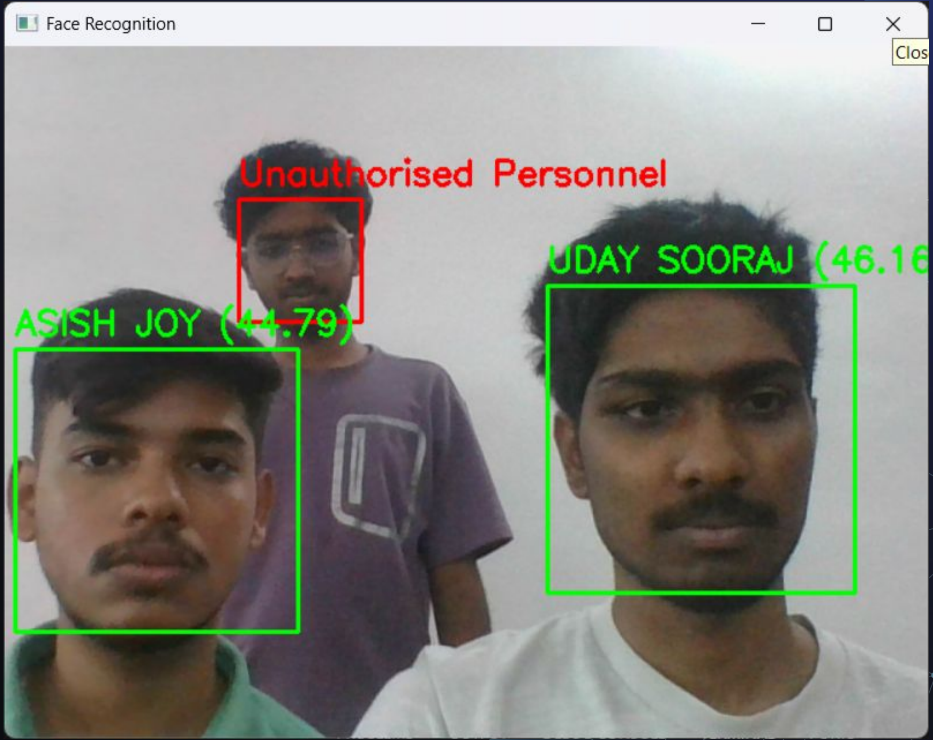
Real-time recognition process



During the recognition process, the trained model is loaded into the system to detect faces in real-time. The model analyzes the captured images and compares them with stored facial data to determine identity. If a recognized face is detected but its confidence level falls below a predefined threshold (e.g., 50), the system considers it as an uncertain match and displays the corresponding name. On the other hand, if the system encounters an unknown face, it immediately triggers an alert mechanism to notify security personnel or take necessary actions. This approach ensures a robust and reliable security framework, minimizing the chances of unauthorized access while maintaining high accuracy in face recognition.

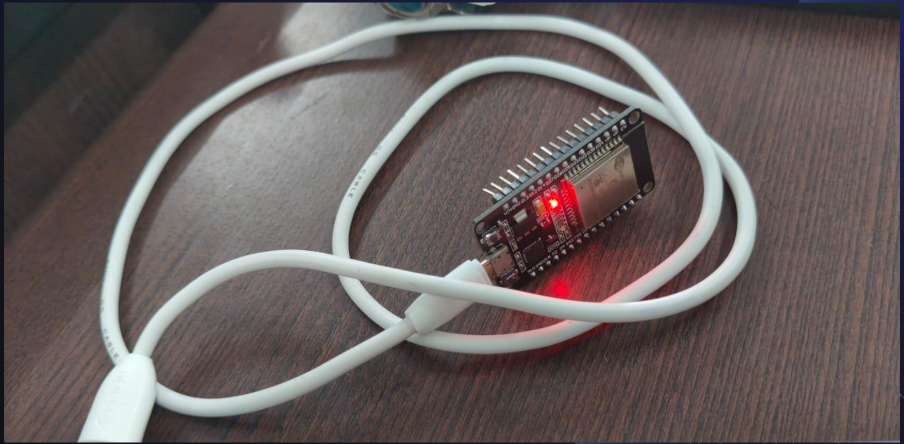
Results


Face Recognition
UDAY SOORAJ (46.26)
ASISH JOY (47.57)

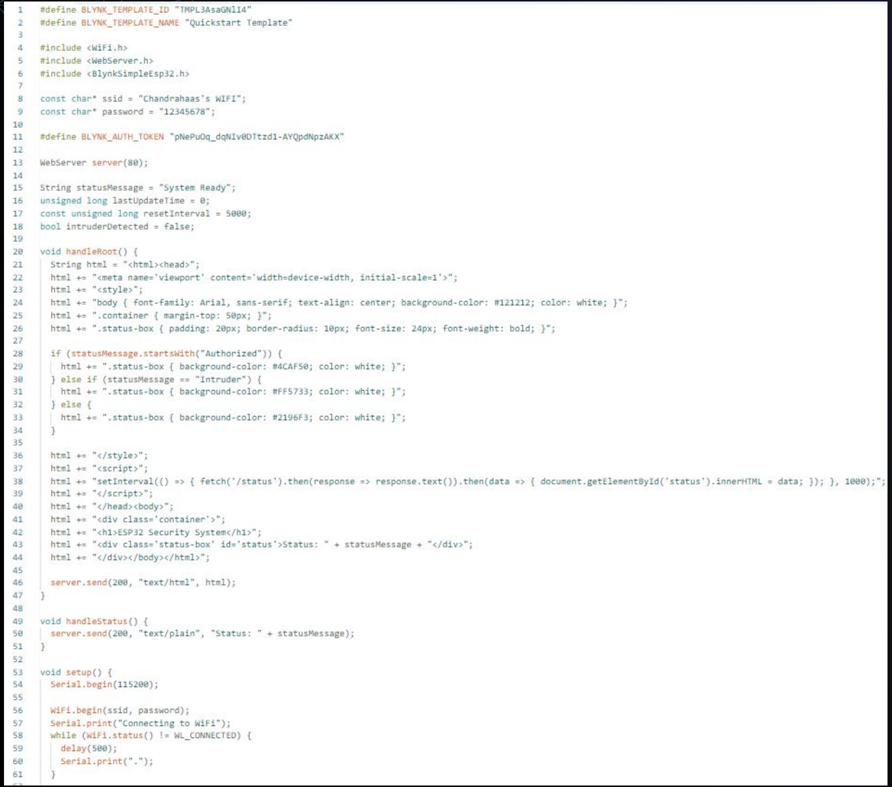

Face Recognition
Clos
Unauthorised Personnel
ASISH JOY (44.79)
UDAY SOORAJ (46.16

ESP32 Serial Communication for status updates

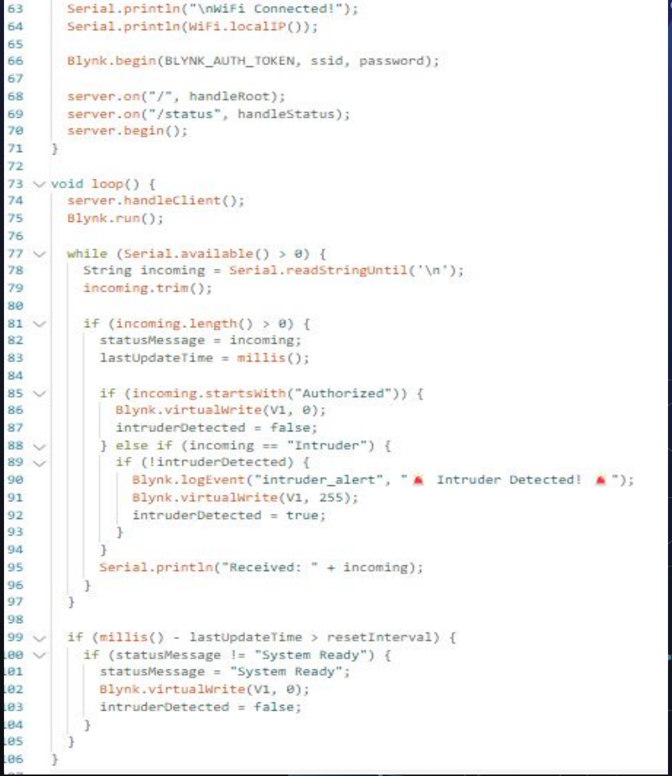
The Python script facilitates communication between the face recognition system and the ESP32 using PySerial, ensuring seamless data transfer. When a face is recognized, the script sends the corresponding identification results to the ESP32, which then updates its web interface to reflect the status of the detected individual. If the person is authorized, the system displays their identity, confirming safe access. Conversely, if the person is unidentified or deemed an intruder, an alert is triggered, notifying users of potential security concerns. This bidirectional communication enables real-time monitoring, allowing for swift responses to unauthorized access attempts and enhancing the overall effectiveness of the security system.



ESP32 Code

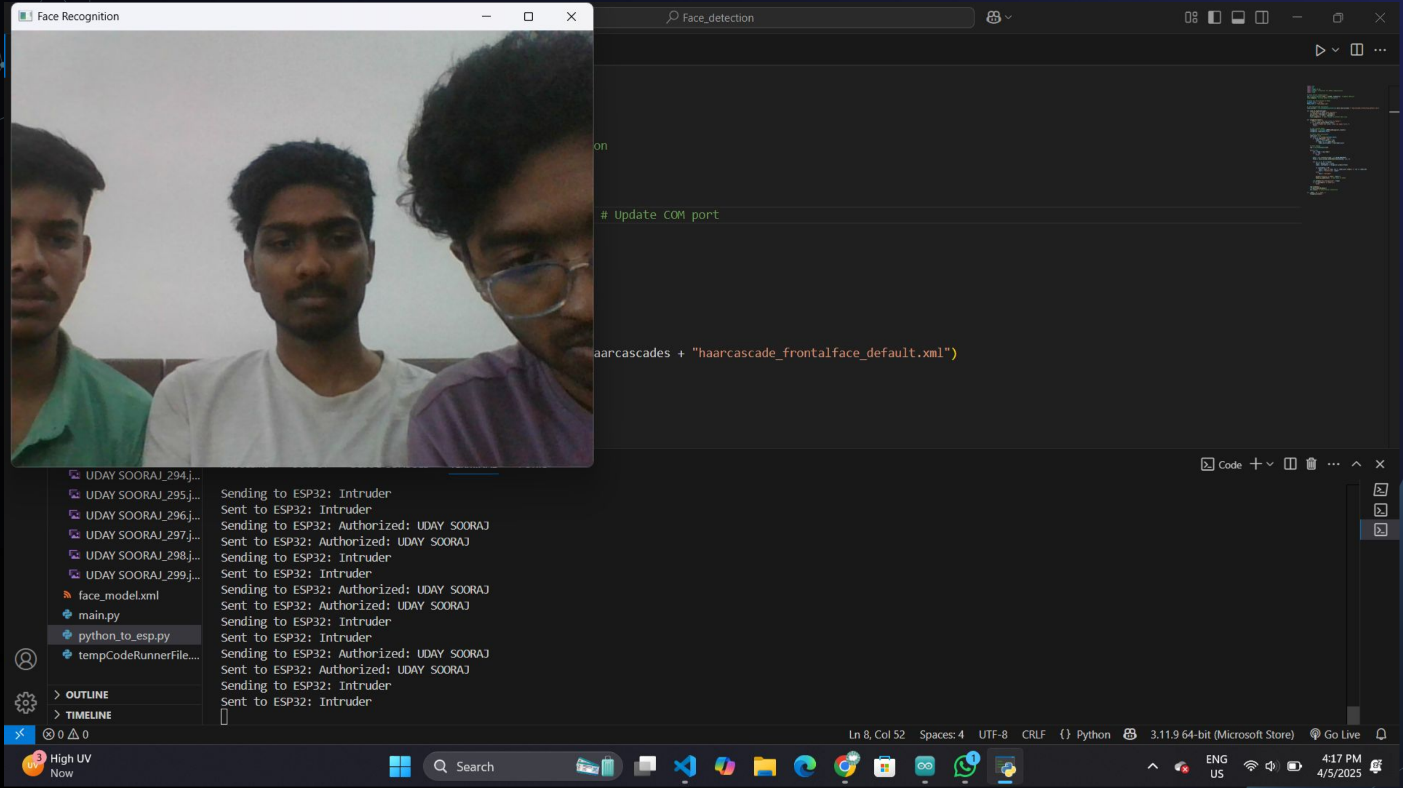


```
#define BLYNK_TEMPLATE_ID "TMPL3AsaGNlI4"
#define BLYNK_TEMPLATE_NAME "Quickstart Template"

#include <WiFi.h>
#include <WebServer.h>
#include <BlynkSimpleEsp32.h>

const char* ssid = "Chandrahaas's WIFI";
const char* password = "12345678";

#define BLYNK_AUTH_TOKEN "pNePuOq_dqNIv0DTtzd1-AYQpdNpzAKX"

WebServer server(80);

String statusMessage = "System Ready";
unsigned long lastUpdateTime = 0;
const unsigned long resetInterval = 5000;
bool intruderDetected = false;

void handleRoot() {
  String html = "<html><head>";
  html += "<meta name='viewport' content='width=device-width, initial-scale=1'>";
  html += "<style>";
  html += "body { font-family: Arial, sans-serif; text-align: center; background-color: #121212; color: white; }";
  html += ".container { margin-top: 50px; }";
  html += ".status-box { padding: 20px; border-radius: 10px; font-size: 24px; font-weight: bold; }";

  if (statusMessage.startsWith("Authorized")) {
    html += ".status-box { background-color: #4CAF50; color: white; }";
  } else if (statusMessage == "Intruder") {
    html += ".status-box { background-color: #FF5733; color: white; }";
  } else {
    html += ".status-box { background-color: #2196F3; color: white; }";
  }

  html += "</style>";
  html += "<script>";
  html += "setInterval(() => { fetch('/status').then(response => response.text()).then(data => { document.getElementById('status').innerHTML = data; }); }, 1000);";
  html += "</script>";
  html += "</head><body>";
  html += "<div class='container'>";
  html += "<h1>ESP32 Security System</h1>";
  html += "<div class='status-box' id='status'>Status: " + statusMessage + "</div>";
  html += "</div></body></html>";

  server.send(200, "text/html", html);
}

void handleStatus() {
  server.send(200, "text/plain", "Status: " + statusMessage);
}

void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```



```
  Serial.println("\nWiFi Connected!");
  Serial.println(WiFi.localIP());

  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, password);

  server.on("/", handleRoot);
  server.on("/status", handleStatus);
  server.begin();
}

void loop() {
  server.handleClient();
  Blynk.run();

  while (Serial.available() > 0) {
    String incoming = Serial.readStringUntil('\n');
    incoming.trim();

    if (incoming.length() > 0) {
      statusMessage = incoming;
      lastUpdateTime = millis();

      if (incoming.startsWith("Authorized")) {
        Blynk.virtualWrite(V1, 0);
        intruderDetected = false;
      } else if (incoming == "Intruder") {
        if (!intruderDetected) {
          Blynk.logEvent("intruder_alert", "🚨 Intruder Detected! 🚨");
          Blynk.virtualWrite(V1, 255);
          intruderDetected = true;
        }
      }
      Serial.println("Received: " + incoming);
    }
  }

  if (millis() - lastUpdateTime > resetInterval) {
    if (statusMessage != "System Ready") {
      statusMessage = "System Ready";
      Blynk.virtualWrite(V1, 0);
      intruderDetected = false;
    }
  }
}
```

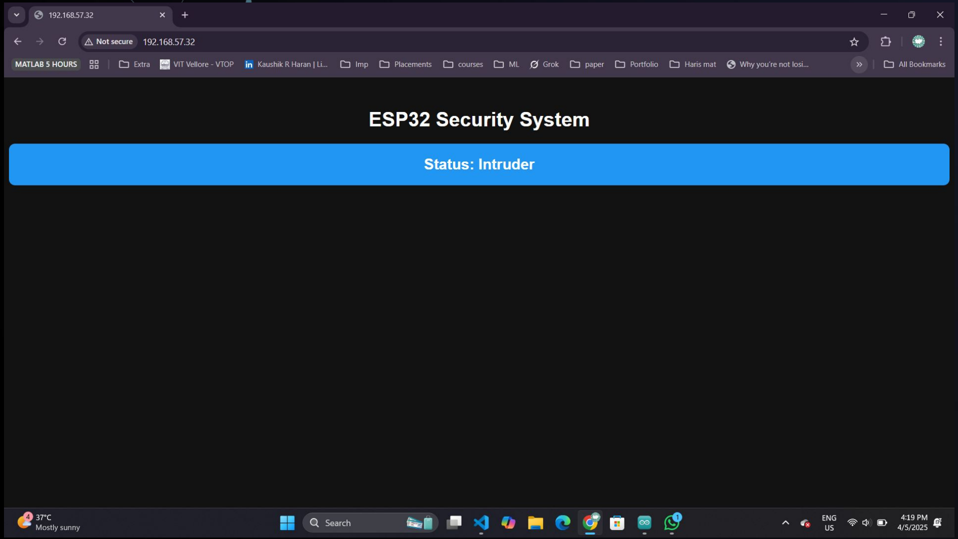
Serial Transmission Results



```
on

# Update COM port

aarcascades + "haarcascade_frontalface_default.xml")
```

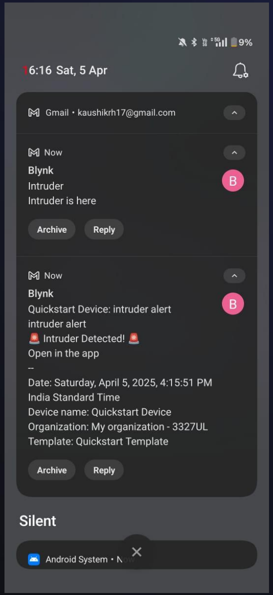
```
Sending to ESP32: Intruder
Sent to ESP32: Intruder
Sending to ESP32: Authorized: UDAY SOORAJ
Sent to ESP32: Authorized: UDAY SOORAJ
Sending to ESP32: Intruder
Sent to ESP32: Intruder
Sending to ESP32: Authorized: UDAY SOORAJ
Sent to ESP32: Authorized: UDAY SOORAJ
Sending to ESP32: Intruder
Sent to ESP32: Intruder
Sending to ESP32: Authorized: UDAY SOORAJ
Sent to ESP32: Authorized: UDAY SOORAJ
Sending to ESP32: Intruder
Sent to ESP32: Intruder
```

Web Server notifications



ESP32 Security System

Status: Intruder



Gmail • kaushikrh17@gmail.com

Now

Blynk

Intruder

Intruder is here

Now

Blynk

Quickstart Device: intruder alert

intruder alert

🚨 Intruder Detected! 🚨

Open in the app

--

Date: Saturday, April 5, 2025, 4:15:51 PM India Standard Time

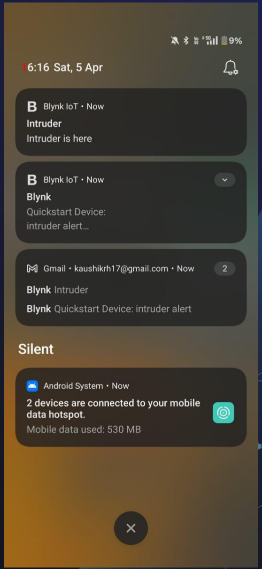
Device name: Quickstart Device

Organization: My organization - 3327UL

Template: Quickstart Template

Silent

Android System • N



Blynk IoT • Now

Intruder

Intruder is here

Blynk IoT • Now

Blynk

Quickstart Device: intruder alert...

Gmail • kaushikrh17@gmail.com • Now 2

Blynk Intruder

Blynk Quickstart Device: intruder alert

Silent

Android System • Now

2 devices are connected to your mobile data hotspot.

Mobile data used: 530 MB

Conclusions

This project successfully combines face recognition technology with the ESP32 microcontroller, yielding an effective security system. With real-time monitoring and alerts, it enhances access control, providing a reliable solution for modern security needs.

Github Link to repository with codes :

Face-Recognition-Based-Security-System