PYTHON MINI PROJECT REPORT

on

" Smart Attendance System using Face Recognition "

By

Pritesh Mahajan - 124A7029

Jalmesh Mhatre - 124A7032

Siddhesh Murkute - 124A7037

SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING

In

Electronics & Computer Science Engineering

DEPARTMENT OF COMPUTER ENGINEERING

SIES GRADUATE SCHOOL OF TECHNOLOGY

NERUL, NAVI MUMBAI – 400706

ACADEMIC YEAR

2025 – 2026

CONTENTS

ABSTRACT

Smart Attendance System through Face Recognition is the latest technology-based solution that has been introduced as a replacement for the traditional methods of attendance like proxy marking, time-consuming and tedious entry work, and the like. This is a computer vision, machine learning-based automated attendance management system implemented in educational institutions and offices. It is coded in Python, using the programming frameworks like OpenCV, along with the LBPH (Local Binary Patterns Histogram) algorithm. It takes real-time videos through the webcam, detects faces via the use of the Haar Cascade classifier, and correctly identifies the registered faces. After the correct identification of the face, the system will automatically register the name, date, and time of the person in a properly structured CSV file, thus saving time on the tedious work of human maintenance. Simple and interactive Tkinter-based GUI helps the users in easily adding new faces, checking attendance entries, and effortlessly running or stopping the attendance process.

This project enhances reliability, efficiency, and security by ensuring that just the authorised individuals are registered as present. It minimises human error, conserves administrative time, and provides a contactless attendance system that is suitable even after the pandemic. The system demonstrates the integration of face recognition technology effectively with everyday applications with the aim of creating smart, automated, and scalable solutions that are suitable for today's institutions.

INTRODUCTION

Accurate reporting of attendance is critical to accountability and future resource allocation, but the ordinary roll-call system is plagued by significant disadvantages. Questionnaires show that teachers spend minutes per lesson marking attendance, and the tedious process is often marred by errors owing to carelessness and fraudulent reporting. In a series of surveys, the laborious schemes have been described as inconvenient, falsifiable, and prone to human error, such that the attendance information can be easily falsified. Those are among the reasons why a trustworthy automated system is essential.

The biometric-based system is a more efficient alternative. Even though fingerprint and smart-card-based systems are widespread, these require physical contact or can be subject to abuse. Conversely, the face recognition system is a contact-free, secure, and instantaneous process of verification. By comparing live facial images with the previously registered database, correct identity is assured, and proxy attendance is eliminated. It has already been put into useful service in schools and offices, becoming quicker and more accurate compared to the traditional method. The proposed system avails itself of these advantages in order to be able to offer efficient and automated attendance management, marking a reduction in administrative work with high accuracy and reliability.

PROPOSED SYSTEM

Proposed System: Automated Face Recognition Attendance System

The proposed system is a desktop application developed in Python that automates the process of recording attendance using facial recognition. It provides a user-friendly graphical interface to manage enrollment and track attendance, effectively replacing manual methods with an efficient, accurate, and digital solution. The system is designed to be modular, separating the core tasks of user interaction, video processing, face recognition, and data storage.

System Architecture and Modules

The application is built upon several key modules that work together to deliver the core functionality.

1. **User Interface (UI) Module:**
    a. **Technology:** Tkinter.
    b. **Function:** Provides the main control window for the user. It features buttons to initiate different modes of operation: starting an attendance session, adding a new user (face enrollment), viewing past attendance records, and exiting the application. It runs the main functions in separate threads to keep the UI responsive.
2. **Face Capture and Processing Module:**
    a. **Technology:** OpenCV.
    b. **Function:** This module is responsible for interfacing with the webcam to capture a live video stream. For each captured frame, it performs essential pre-processing steps, such as converting the image to grayscale (which is sufficient and faster for recognition) and mirroring the frame for a more intuitive user experience.

3. **Face Detection Module:**
    a. **Technology:** OpenCV's Haar Cascades.

b. **Function:** It uses a pre-trained Haar Cascade classifier (`haarcascade_frontalface_default.xml`) to detect and locate human faces within each video frame. This process identifies the coordinates of a bounding box around each detected face.

4. **Face Recognition Module:**
    a. **Technology:** OpenCV's LBPH (Local Binary Patterns Histograms) Face Recognizer.
    b. **Function:** This is the core of the system's intelligence.
        i. **Training:** The LBPH algorithm is trained on a collection of images of known individuals stored in the `known_faces` directory. It learns to associate facial features with a unique numerical label for each person.
        ii. **Prediction:** During the attendance session, the module takes the detected face region, resizes it to a standard size (200×200 pixels), and uses the trained recognizer to predict the person's identity. It returns a label and a confidence score (distance), where a lower score indicates a better match.
5. **Data Storage Module:**
    a. **Technology:** File System (OS) and CSV Module.
    b. **Function:** This module handles all data persistence.
        i. **Known Faces Database:** It manages a directory named `known_faces`, where a grayscale image for each registered person is stored. This acts as the visual database for training the recognizer.
        ii. **Attendance Log:** It maintains an `attendance.csv` file. When a person is successfully recognized, their name, the current date, and the timestamp are appended as a new row to this file.

SYSTEM WORKFLOW

The system operates in two primary workflows: **Enrollment (Adding a New Face)** and **Attendance Marking**.

1. Enrollment Workflow

This process is used to register a new person into the system's database.

1. The user clicks the **"Add New Face"** button in the main window.
2. The system prompts the user to enter the person's name via the command line.
3. A new window opens, displaying the live webcam feed. A green rectangle is drawn around any detected face.
4. The user positions their face correctly and presses the **SPACE bar** to capture the image.
5. The system isolates the detected face region, converts it to grayscale, resizes it, and saves it as a new `.jpg` file in the `known_faces` directory. The filename includes the person's name and a timestamp to ensure uniqueness.
6. Finally, the `load_known_faces()` function is called again to **retrain the LBPH recognizer** with the newly added face, making it immediately available for recognition.

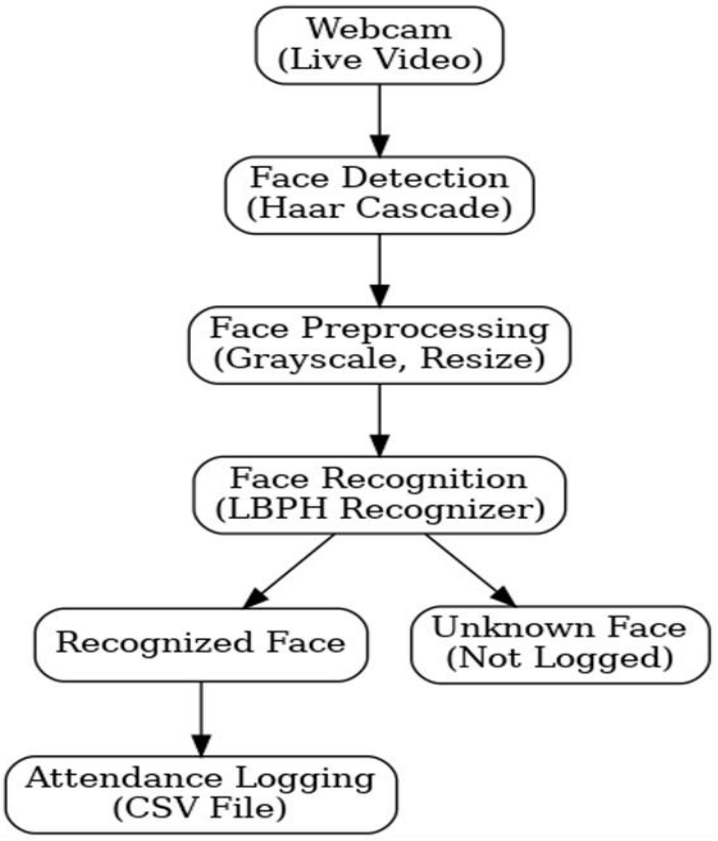
2. Attendance Marking Workflow

This is the main operational mode of the application.

1. The user clicks the **"Start Attendance"** button.
2. A window appears showing the live, mirrored webcam feed.
3. In a continuous loop, the system performs the following for each frame:
    a. Detects all faces in the frame using Haar Cascades.
    b. For each detected face, it extracts the Region of Interest (ROI).
    c. The face ROI is passed to the trained **LBPH recognizer**, which predicts the person's identity (label) and a confidence score.
    d. If the confidence score is below the predefined threshold (`RECOG_THRESHOLD`), the match is considered valid. The system retrieves the person's name corresponding to the predicted label.

e. A check is performed to see if this person has already been marked in the current session (`session_marked` set) to prevent duplicate entries.
f. If it's a new, valid recognition for the session, the system calls the `mark_attendance()` function to append the **person's name, date, and time** to the `attendance.csv` file.
g. A bounding box and the person's name (in green for recognized, red for "Unknown") are drawn on the video feed to provide real-time visual feedback.

4. The user can press the **ESC key** to close the camera window and end the session.



PROGRAM / CODE

```python
import os
import csv
from datetime import datetime
import cv2
import numpy as np
import threading
import tkinter as tk
from tkinter import messagebox, scrolledtext

# ---------- Config ----------
KNOWN_DIR = "known_faces"
ATTENDANCE_CSV = "attendance.csv"
FACE_SIZE = (200, 200)
RECOG_THRESHOLD = 60  # Lower = stricter match
# ----------------------------

face_cascade = None
recognizer = None
FACE_MODULE_AVAILABLE = False

known_faces = []
labels = []
label_to_name = {}  # Correct mapping

# ---------- Initialization ----------
def initialize_modules():
    global face_cascade, recognizer, FACE_MODULE_AVAILABLE
    try:
        haar_path = os.path.join(os.path.dirname(cv2.__file__), "data", "haarcascade_frontalface_default.xml")
        face_cascade = cv2.CascadeClassifier(haar_path)
        if face_cascade.empty():
            face_cascade = None
            print("⚠ Haar cascade not found.")
    except Exception as e:
        print("Cascade load error:", e)

```

```python
    try:
        face_module = getattr(cv2, "face", None)
        if face_module is not None:
            recognizer_local = face_module.LBPHFaceRecognizer_create()
            recognizer_local.setThreshold(RECOG_THRESHOLD)
            global recognizer
            recognizer = recognizer_local
            FACE_MODULE_AVAILABLE = True
        else:
            print("⚠ cv2.face not available (install opencv-contrib-python).")
    except Exception as e:
        print("Face recognizer init error:", e)

def ensure_known_dir():
    if not os.path.exists(KNOWN_DIR):
        os.makedirs(KNOWN_DIR)

def load_known_faces():
    """Load faces and train recognizer once, create deterministic label-to-name mapping"""
    global known_faces, labels, label_to_name, recognizer
    ensure_known_dir()
    files = [f for f in os.listdir(KNOWN_DIR) if f.lower().endswith(('.jpg','.jpeg','.png'))]
    if not files:
        print("No faces found in known_faces folder.")
        return

    known_faces = []
    labels = []
    name_to_label = {}
    next_label = 0

    for fname in files:
        path = os.path.join(KNOWN_DIR, fname)
        img = cv2.imread(path)
        if img is None:
            continue
```

```
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray, 1.1, 4, minSize=(30,30)) if face_cascade else []
            face = gray
            if len(faces) > 0:
                x, y, w, h = faces[0]
                face = gray[y:y+h, x:x+w]
            try:
                face_resized = cv2.resize(face, FACE_SIZE)
            except:
                continue
            # Extract name before underscore or extension
            name = os.path.splitext(fname)[0].split("_")[0]
            if name not in name_to_label:
                name_to_label[name] = next_label
                label_to_name[next_label] = name
                next_label += 1
            label = name_to_label[name]

            known_faces.append(face_resized)
            labels.append(label)
            print(f"Loaded face: {name}")

    # Train recognizer
    if FACE_MODULE_AVAILABLE and recognizer is not None and known_faces:
        recognizer.train(known_faces, np.array(labels))
        print(f"Recognizer trained with {len(label_to_name)} unique faces.")

# ---------- Attendance ----------
def initialize_attendance_file():
    if not os.path.isfile(ATTENDANCE_CSV):
        with open(ATTENDANCE_CSV, 'w', newline='') as f:
            csv.writer(f).writerow(['Name','Date','Time'])

def mark_attendance(name):
    now = datetime.now()
    date_s = now.strftime("%Y-%m-%d")
```

```
    time_s = now.strftime("%H:%M:%S")
    with open(ATTENDANCE_CSV, 'a', newline='', encoding='utf-8') as f:
        csv.writer(f).writerow([name, date_s, time_s])
    print(f"✅ Marked attendance: {name} at {date_s} {time_s}")

def get_attendance_records():
    if not os.path.isfile(ATTENDANCE_CSV):
        return []
    with open(ATTENDANCE_CSV, 'r') as f:
        return list(csv.reader(f))

def save_new_face(image_bgr, name):
    ensure_known_dir()
    gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
    try:
        gray = cv2.resize(gray, FACE_SIZE)
    except:
        return False
    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")
    filename = os.path.join(KNOWN_DIR, f"{name}_{timestamp}.jpg")
    success = cv2.imwrite(filename, gray)
    if success:
        print(f"Saved new face: {filename}")
    return success

# ---------- Camera ----------
def open_camera():
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("❌ Cannot open camera.")
        return None
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
    cap.set(cv2.CAP_PROP_FPS, 30)
    return cap
```

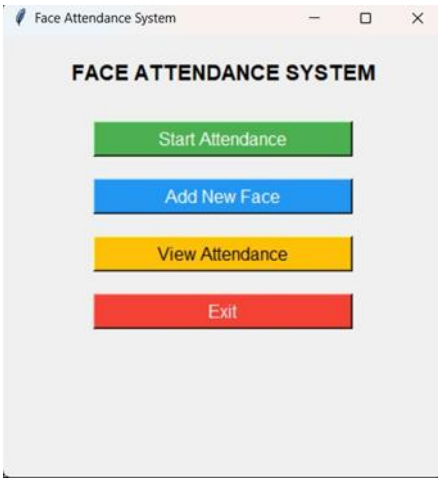
```python
# ---------- Modes ----------
def add_new_face():
    name = input("Enter name for new face: ").strip()
    if not name:
        print("Name cannot be empty.")
        return
    cap = open_camera()
    if cap is None:
        return
    print("\n[Press SPACE to capture, ESC to cancel]")
    while True:
        ret, frame = cap.read()
        if not ret:
            continue
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.1, 4, minSize=(30,30)) if face_cascade else []
        for (x,y,w,h) in faces:
            cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)
        cv2.imshow("Add Face", frame)
        key = cv2.waitKey(1)
        if key == 27:  # ESC
            break
        elif key == 32:  # SPACE
            if len(faces) == 0:
                print("No face detected, try again.")
                continue
            x, y, w, h = faces[0]
            face_roi = frame[y:y+h, x:x+w]
            if save_new_face(face_roi, name):
                load_known_faces()
                print(f"✅ Face saved for {name}")
            break
    cap.release()
    cv2.destroyAllWindows()

def start_attendance():
```

```python
    cap = open_camera()
    if cap is None:
        return
    session_marked = set()
    print("\n[Press ESC to exit attendance mode]")
    while True:
        ret, frame = cap.read()
        if not ret:
            continue
        mirrored = cv2.flip(frame,1)
        gray = cv2.cvtColor(mirrored, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray,1.1,4,minSize=(30,30)) if face_cascade else []
        for (x,y,w,h) in faces[:3]:
            face_roi = gray[y:y+h, x:x+w]
            face_resized = cv2.resize(face_roi, FACE_SIZE)
            label_text = "Unknown"
            color = (0,0,255)
            if FACE_MODULE_AVAILABLE and recognizer is not None and known_faces:
                try:
                    label, confidence = recognizer.predict(face_resized)
                    if confidence <= RECOG_THRESHOLD:
                        name = label_to_name.get(label,"Unknown")
                        label_text = name
                        color = (0,255,0)
                        if name not in session_marked:
                            mark_attendance(name)
                            session_marked.add(name)
                    else:
                        label_text = "Unknown"
                except:
                    label_text = "Error"
            cv2.rectangle(mirrored,(x,y),(x+w,y+h),color,2)
            cv2.putText(mirrored,label_text,(x,y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,color,2)
        cv2.imshow("Attendance Mode",mirrored)
        if cv2.waitKey(1) == 27:
            break
```

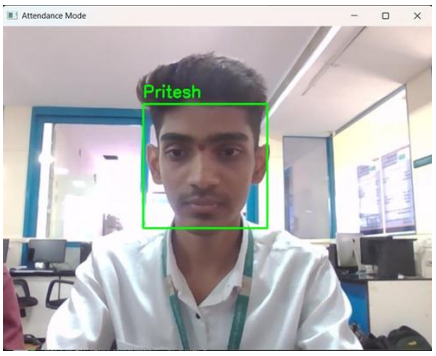
```
    cap.release()
    cv2.destroyAllWindows()

# ---------- GUI ----------
def run_in_thread(func):
    t = threading.Thread(target=func)
    t.daemon = True
    t.start()

def show_attendance_window():
    records = get_attendance_records()
    if len(records) <= 1:
        messagebox.showinfo("Attendance","No attendance data yet.")
        return
    top = tk.Toplevel()
    top.title("Attendance Records")
    top.geometry("400x300")
    text = scrolledtext.ScrolledText(top, wrap=tk.WORD, font=("Courier New",10))
    text.pack(fill=tk.BOTH,expand=True)
    text.insert(tk.END,"Name                Date         Time\n")
    text.insert(tk.END,"-"*35 + "\n")
    for row in records[1:]:
        if len(row)>=3:
            text.insert(tk.END,f"{row[0]:<20} {row[1]:<12} {row[2]:<10}\n")
    text.config(state=tk.DISABLED)

def gui_main():
    initialize_modules()
    load_known_faces()
    initialize_attendance_file()
    root = tk.Tk()
    root.title("Face Attendance System")
    root.geometry("400x400")
    root.config(bg="#f0f0f0")
    tk.Label(root,text="FACE ATTENDANCE SYSTEM",font=("Arial",14,"bold"),bg="#f0f0f0").pack(pady=20)
    tk.Button(root,text="Start Attendance",bg="#4CAF50",fg="white",font=("Arial",12),
```

```
              width=25, command=lambda: run_in_thread(start_attendance)).pack(pady=10)
    tk.Button(root,text="Add New Face",bg="#2196F3",fg="white",font=("Arial",12),
              width=25, command=lambda: run_in_thread(add_new_face)).pack(pady=10)
    tk.Button(root,text="View Attendance",bg="#FFC107",fg="black",font=("Arial",12),
              width=25, command=show_attendance_window).pack(pady=10)
    tk.Button(root,text="Exit",bg="#f44336",fg="white",font=("Arial",12),
              width=25, command=root.destroy).pack(pady=10)
    root.mainloop()

if __name__ == "__main__":
    gui_main()
```

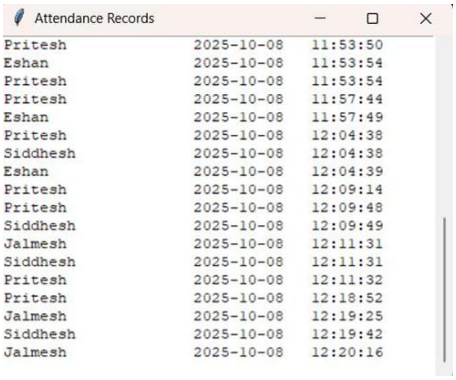
SNAPSHOTS OF WORKING PROJECT



The Smart Attendance System provides a clear and user-friendly interface designed using Tkinter, ensuring smooth interaction for the user. The main **GUI window** serves as the control hub, offering options to Start Attendance, Add New Face, View Attendance, and *Exit*. Through this interface, users can easily perform all operations — from capturing new faces to viewing recorded attendance — without requiring any technical expertise.



During operation, the Face Detection and Recognition module uses the webcam to capture live video feeds. The Haar Cascade Classifier detects faces in real time, and the LBPH (Local Binary Patterns Histogram) recognizer identifies each individual by matching the detected face with stored facial data. Once identified, the recognized person's name is displayed on the video screen inside a bounding box, providing instant feedback to the user.



After successful recognition, the system automatically updates the Attendance Log (CSV Output). Each record includes the individual's name, date, and timestamp of recognition, all saved securely in a CSV file for future reference. This ensures that attendance data remains accurate, easily retrievable, and free from manual entry errors. Overall, the interface integrates data capture, recognition, and record management seamlessly, providing an efficient and modern attendance tracking experience.

CONCLUSION & FUTURE SCOPE

In conclusion, this project provides a strong and effective prototype for a smart attendance solution. It successfully demonstrates the feasibility of using computer vision to overcome the chronic inefficiencies of traditional attendance methods. By directly addressing issues like time consumption, human error, and proxy attendance, the system delivers an immediate and tangible improvement in operational workflow. The key benefit it offers is the creation of an automatic, accurate, and auditable digital log, which enhances accountability and saves valuable administrative time.

While it stands as a robust proof-of-concept, its true value lies in its foundational architecture. This prototype is not merely an endpoint but a launchpad for developing more sophisticated, enterprise-grade attendance management systems. It establishes the core logic of face capture, recognition, and logging, upon which the advanced features outlined in the future scope can be seamlessly integrated. The project confirms that even a lightweight, desktop-based solution can bring significant reliability and efficiency to the critical task of attendance tracking.

Future Scope:

- **Advanced Recognition Models:** Integrate deep learning models like CNNs or FaceNet for better accuracy in poor lighting or with face masks.
- **Multi-Face Recognition:** Upgrade the system to identify multiple faces simultaneously in a single frame for faster processing in groups.
- **Cloud and Mobile Integration:** Develop a cloud backend and a mobile app for remote access to real-time attendance data and alerts.
- **Database Integration:** Create APIs to connect with existing Student Information Systems (SIS) or employee databases for automatic record syncing.
- **Analytics and Alerts:** Implement a dashboard for data insights and configure real-time alerts for events like prolonged absenteeism.

REFRENCES

1. OpenCV Documentation

Website: https://opencv.org

Description: Comprehensive documentation and tutorials on the OpenCV library, used for image processing and computer vision applications.

2. Python Official Documentation

Website: https://www.python.org

Description: Official resources and standard library documentation for the Python programming language, used for coding and logic implementation in this project.

3. Project README File (Face Detection Attendance Recognition)

Description: Detailed project setup guide and implementation steps prepared during development, outlining module structure and usage instructions.

4. GeeksforGeeks – Tutorials on OpenCV & Tkinter

Website: https://www.geeksforgeeks.org

Description: Educational tutorials on computer vision and GUI development.

5. TutorialsPoint – Python GUI Programming & Face Recognition Resources

Website: https://www.tutorialspoint.com

Description: Step-by-step guides for Python GUI and face recognition implementation.