

Name: Adeel Ali Yousaf Class: BS Computer Science

Roll no. 24010102 – 011 Section: 6th Blue

Lecturer: Sir Osama Waheed Subject: Artificial Intelligence Lab

Lab Project: Face Recognition Attendance System

Code:

1. Attendance.py:

import tkinter as tk

from tkinter import messagebox

from database import Database

import cv2

import face\_recognition

import numpy as np

import os

import pickle

import datetime

import pandas as pd

from face\_recognition\_module import get\_encodings\_file

class StudentAttendance:

    def \_\_init\_\_(self, root, session\_start):

        self.root = root

        self.root.title("Student Attendance - Face Recognition")

        self.session\_start = session\_start

        self.frame = tk.Frame(self.root, padx=20, pady=20)

        self.frame.pack()

        tk.Label(self.frame, text="Student Attendance", font=("Arial", 16)).pack(pady=10)

        self.status\_label = tk.Label(self.frame, text="", font=("Arial", 12))

        self.status\_label.pack(pady=10)

        tk.Button(self.frame, text="Mark Attendance", width=20, command=self.mark\_attendance).pack(pady=10)

        tk.Button(self.frame, text="Export Attendance", width=20, command=self.export\_attendance\_to\_excel).pack(pady=10)

    def export\_attendance\_to\_excel(self):

        db = Database()

        try:

            query = """

                SELECT s.name, s.class, s.roll\_no, a.timestamp

                FROM attendance a

                JOIN student s ON a.student\_id = s.id

                WHERE a.timestamp >= %s

                ORDER BY a.timestamp

            """

            records = db.fetch(query, (self.session\_start,))

            if not records:

                messagebox.showinfo("Attendance Export", "No attendance records found for this session.")

                return

            df = pd.DataFrame(records, columns=["Name", "Class", "Roll Number", "Timestamp"])

            filename = f"attendance\_{datetime.datetime.now().strftime('%Y%m%d\_%H%M%S')}.xlsx"

            df.to\_excel(filename, index=False)

            messagebox.showinfo("Attendance Export", f"Attendance exported to {filename}")

        except Exception as e:

            messagebox.showerror("Export Error", f"Failed to export attendance.\n{e}")

        finally:

            db.close()

    def mark\_attendance(self):

        encoding\_file = get\_encodings\_file("student")

        if not os.path.exists(encoding\_file):

            messagebox.showerror("Error", "No registered students found.")

            return

        with open(encoding\_file, "rb") as f:

            known\_faces = pickle.load(f)

        known\_encodings = [entry["encoding"] for entry in known\_faces]

        known\_ids = [entry["id"] for entry in known\_faces]

        known\_names = [entry["name"] for entry in known\_faces]

        cam = cv2.VideoCapture(0)

        intruder\_dir = "intruder"

        os.makedirs(intruder\_dir, exist\_ok=True)

        recognized\_id = None

        recognized\_name = None

        while True:

            ret, frame = cam.read()

            if not ret:

                break

            rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

            boxes = face\_recognition.face\_locations(rgb\_frame)

            encodings = face\_recognition.face\_encodings(rgb\_frame, boxes)

            if boxes:

                encoding = encodings[0]

                matches = face\_recognition.compare\_faces(known\_encodings, encoding)

                distances = face\_recognition.face\_distance(known\_encodings, encoding)

                best\_match\_index = np.argmin(distances)

                if matches[best\_match\_index] and distances[best\_match\_index] < 0.5:

                    recognized\_id = known\_ids[best\_match\_index]

                    recognized\_name = known\_names[best\_match\_index]

                    # Display recognized name

                    cv2.putText(frame, f"Recognized: {recognized\_name}", (10, 30),

                                cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

                    cv2.imshow("Face Recognition", frame)

                    cv2.waitKey(1000)

                    break

                else:

                    # Draw red rectangle for intruder

                    top, right, bottom, left = boxes[0]

                    cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 3)

                    timestamp = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S")

                    cv2.imwrite(f"{intruder\_dir}/intruder\_{timestamp}.jpg", frame)

                    cv2.imshow("Face Recognition - Intruder", frame)

                    cv2.waitKey(2000)

                    break

            else:

                cv2.imshow("Face Recognition", frame)

            if cv2.waitKey(1) & 0xFF == ord("q"):

                break

        cam.release()

        cv2.destroyAllWindows()

        if recognized\_id:

            db = Database()

            try:

                db.execute("INSERT INTO attendance (student\_id) VALUES (%s)", (recognized\_id,))

                self.status\_label.config(text=f"✔️ {recognized\_name}'s attendance marked.", fg="green")

            except Exception as e:

                messagebox.showerror("Database Error", f"Could not mark attendance.\n{e}")

            finally:

                db.close()

        else:

            if not self.status\_label.cget("text"):

                self.status\_label.config(text="❌ Face not recognized.", fg="red")

1. Database.py:  
   import mysql.connector

class Database:

    def \_\_init\_\_(self):

        self.conn = mysql.connector.connect(

            host="localhost",

            user="root",

            password="",

            database="ailabproject"

        )

        self.cursor = self.conn.cursor()

    def execute(self, query, params=None):

        self.cursor.execute(query, params or ())

        self.conn.commit()

    def fetch(self, query, params=None):

        self.cursor.execute(query, params or ())

        return self.cursor.fetchall()

    def close(self):

        self.cursor.close()

        self.conn.close()

1. face\_recognition\_module.py:

import face\_recognition

import cv2

import os

import numpy as np

import pickle

import uuid

DATASET\_DIR = "dataset/"

ENCODINGS\_FILE = "trainer/encodings.pkl"

def capture\_and\_encode(student\_id, name):

    cam = cv2.VideoCapture(0)

    count = 0

    face\_encodings = []

    student\_dir = os.path.join(DATASET\_DIR, f"{student\_id}\_{name}")

    os.makedirs(student\_dir, exist\_ok=True)

    while count < 10:

        ret, frame = cam.read()

        rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        boxes = face\_recognition.face\_locations(rgb\_frame)

        if boxes:

            encodings = face\_recognition.face\_encodings(rgb\_frame, boxes)

            if encodings:

                face\_encodings.append(encodings[0])

                count += 1

                # Save a frame for reference

                cv2.imwrite(os.path.join(student\_dir, f"{count}.jpg"), frame)

                cv2.rectangle(frame, boxes[0][::-1][0:2], boxes[0][::-1][2:], (0, 255, 0), 2)

        cv2.imshow("Capturing Face...", frame)

        if cv2.waitKey(1) == 27:

            break

    cam.release()

    cv2.destroyAllWindows()

    if face\_encodings:

        save\_encoding(student\_id, name, face\_encodings)

        return True

    return False

def save\_encoding(student\_id, name, encodings):

    # Ensure trainer directory exists

    os.makedirs(os.path.dirname(ENCODINGS\_FILE), exist\_ok=True)

    if os.path.exists(ENCODINGS\_FILE):

        with open(ENCODINGS\_FILE, "rb") as f:

            known = pickle.load(f)

    else:

        known = []

    for encoding in encodings:

        known.append({

            "id": student\_id,

            "name": name,

            "encoding": encoding

        })

    with open(ENCODINGS\_FILE, "wb") as f:

        pickle.dump(known, f)

def recognize\_face(user\_type="student"):

    encoding\_file = get\_encodings\_file(user\_type)

    if not os.path.exists(encoding\_file):

        return None

    with open(encoding\_file, "rb") as f:

        known\_faces = pickle.load(f)

    known\_encodings = [entry["encoding"] for entry in known\_faces]

    known\_ids = [entry["id"] for entry in known\_faces]

    cam = cv2.VideoCapture(0)

    recognized\_id = None

    while True:

        ret, frame = cam.read()

        rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        boxes = face\_recognition.face\_locations(rgb\_frame)

        encodings = face\_recognition.face\_encodings(rgb\_frame, boxes)

        for encoding in encodings:

            matches = face\_recognition.compare\_faces(known\_encodings, encoding)

            distances = face\_recognition.face\_distance(known\_encodings, encoding)

            best\_match\_index = np.argmin(distances)

            if matches[best\_match\_index] and distances[best\_match\_index] < 0.5:

                recognized\_id = known\_ids[best\_match\_index]

                cam.release()

                cv2.destroyAllWindows()

                return recognized\_id

        cv2.imshow("Recognizing Face...", frame)

        if cv2.waitKey(1) & 0xFF == ord("q"):

            break

    cam.release()

    cv2.destroyAllWindows()

    return None

def get\_encodings\_file(user\_type):

    return f"trainer/encodings\_{user\_type}.pkl"

def register\_face(user\_type="teacher", name=""):

    user\_id = str(uuid.uuid4())  # Generate a unique ID

    cam = cv2.VideoCapture(0)

    count = 0

    face\_encodings = []

    user\_dir = os.path.join(DATASET\_DIR, f"{user\_id}\_{name}")

    os.makedirs(user\_dir, exist\_ok=True)

    while count < 10:

        ret, frame = cam.read()

        rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        boxes = face\_recognition.face\_locations(rgb\_frame)

        if boxes:

            encodings = face\_recognition.face\_encodings(rgb\_frame, boxes)

            if encodings:

                face\_encodings.append(encodings[0])

                count += 1

                cv2.imwrite(os.path.join(user\_dir, f"{count}.jpg"), frame)

                cv2.rectangle(frame, boxes[0][::-1][0:2], boxes[0][::-1][2:], (0, 255, 0), 2)

        cv2.imshow(f"Registering {user\_type}...", frame)

        if cv2.waitKey(1) == 27:  # ESC to cancel

            break

    cam.release()

    cv2.destroyAllWindows()

    if face\_encodings:

        encoding\_file = get\_encodings\_file(user\_type)

        # Ensure trainer directory exists

        os.makedirs(os.path.dirname(encoding\_file), exist\_ok=True)

        if os.path.exists(encoding\_file):

            with open(encoding\_file, "rb") as f:

                known = pickle.load(f)

        else:

            known = []

        for encoding in face\_encodings:

            known.append({

                "id": user\_id,

                "name": name,

                "encoding": encoding

            })

        with open(encoding\_file, "wb") as f:

            pickle.dump(known, f)

        return True, user\_id

    return False, None

1. login.py:  
   import tkinter as tk

from tkinter import messagebox, simpledialog

from face\_recognition\_module import recognize\_face, register\_face

from database import Database

from main\_menu import MainMenu

from datetime import datetime

class Login:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Teacher Login - Face Recognition")

        self.login\_frame = tk.Frame(self.root)

        self.login\_frame.pack(pady=50)

        self.login\_time=None

        tk.Label(self.login\_frame, text="Login Using Face Recognition", font=("Arial", 14)).pack(pady=20)

        tk.Button(self.login\_frame, text="Login", width=20, command=self.authenticate).pack(pady=10)

        tk.Button(self.login\_frame, text="Register", width=20, command=self.register).pack(pady=10)

    def authenticate(self):

        teacher\_id = recognize\_face("teacher")

        if teacher\_id:

            db = Database()

            teacher = db.fetch("SELECT name, class FROM teacher WHERE id=%s", (teacher\_id,))

            db.close()

            if teacher:

                self.login\_time = datetime.now()

                name, class\_ = teacher[0]

                self.login\_frame.destroy()

                MainMenu(self.root, name, class\_, self.login\_time)

            else:

                messagebox.showerror("Error", "Teacher not found in database.")

        else:

            messagebox.showerror("Error", "Face not recognized.")

    def register(self):

        name = simpledialog.askstring("Input", "Enter Teacher Name:", parent=self.root)

        if not name:

            return

        class\_ = simpledialog.askstring("Input", "Enter Class Name:", parent=self.root)

        if not class\_:

            return

        success, teacher\_id = register\_face("teacher", name)

        if success:

            db = Database()

            db.execute("INSERT INTO teacher (id, name, class) VALUES (%s, %s, %s)", (teacher\_id, name, class\_))

            db.close()

            messagebox.showinfo("Success", f"Registered {name} successfully.")

        else:

            messagebox.showerror("Error", "Face registration failed.")

1. main\_menu.py:  
   import tkinter as tk

from student\_admission import StudentAdmission

from attendance import StudentAttendance

class MainMenu:

    def \_\_init\_\_(self, root, teacher\_name, teacher\_class, session\_start):

        self.root = root

        self.menu\_frame = tk.Frame(self.root)

        self.menu\_frame.pack()

        self.session\_start = session\_start

        tk.Label(self.menu\_frame, text=f"Welcome {teacher\_name} - Class: {teacher\_class}").pack(pady=20)

        tk.Button(self.menu\_frame, text="Student Admission", command=self.student\_admission).pack(pady=10)

        tk.Button(self.menu\_frame, text="Mark Attendance", command=self.student\_attendance).pack(pady=10)

    def student\_admission(self):

        self.menu\_frame.destroy()

        StudentAdmission(self.root)

    def student\_attendance(self):

        self.menu\_frame.destroy()

        StudentAttendance(self.root, self.session\_start)

1. Student\_admission.py:

import tkinter as tk

from tkinter import messagebox

from database import Database

from face\_recognition\_module import capture\_and\_encode

class StudentAdmission:

    def \_\_init\_\_(self, root):

        self.root = root

        self.frame = tk.Frame(self.root)

        self.frame.pack()

        self.name\_var = tk.StringVar()

        self.roll\_var = tk.StringVar()

        self.class\_var = tk.StringVar()

        tk.Label(self.frame, text="Student Admission Form").pack(pady=20)

        tk.Label(self.frame, text="Name").pack()

        tk.Entry(self.frame, textvariable=self.name\_var).pack()

        tk.Label(self.frame, text="Roll No").pack()

        tk.Entry(self.frame, textvariable=self.roll\_var).pack()

        tk.Label(self.frame, text="Class").pack()

        tk.Entry(self.frame, textvariable=self.class\_var).pack()

        tk.Button(self.frame, text="Capture Face", command=self.capture\_face).pack(pady=10)

    def capture\_face(self):

        name = self.name\_var.get()

        roll = self.roll\_var.get()

        class\_ = self.class\_var.get()

        if not name or not roll or not class\_:

            messagebox.showerror("Error", "All fields are required")

            return

        db = Database()

        db.execute("INSERT INTO student (name, roll\_no, class) VALUES (%s, %s, %s)", (name, roll, class\_))

        student\_id = db.cursor.lastrowid

        db.close()

        success = capture\_and\_encode(student\_id, name)

        if success:

            messagebox.showinfo("Success", "Face Captured and Encoded Successfully")

        else:

            messagebox.showerror("Error", "Failed to capture a valid face")

1. main.py:  
   import tkinter as tk

from login import Login

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    root.geometry("400x300")

    app = Login(root)

    root.mainloop()

Output:









