Face Recognition Attendance System

Introduction

This document explains the functionality, features, and code implementation of the Face Recognition Attendance System. This system uses OpenCV and face_recognition libraries to detect and recognize multiple faces simultaneously and mark attendance in an Excel file.

Features

- ✔ Register new faces with unique names.
- ✔ Prevent duplicate face registrations.
- ✔ Mark attendance for multiple faces at the same time.
- ✔ Real-time attendance display.
- ✔ Delete registered persons.
- ✔ View attendance records of the last 24 hours.
- ✔ Show all registered persons in the system.

Code Explanation

The system consists of several functions that perform face registration, face recognition, attendance marking, and user interactions via a Tkinter GUI.

1. Load Known Faces

This function loads all registered faces from the 'faces' directory and stores their encodings for comparison.

2. Register New Faces

Captures a face from the camera, checks if it is already registered, and saves it with a user-defined name.

3. Mark Attendance

Marks attendance in an Excel sheet by storing the name and timestamp.

4. Face Recognition

Detects and recognizes multiple faces in real-time and updates attendance accordingly.

5. GUI Implementation

Uses Tkinter to provide buttons for different functionalities like registration, attendance marking, and deleting persons.

Complete Code

```
import cv2
import face_recognition
import tkinter as tk
```

```
from tkinter import simpledialog, messagebox, Toplevel
import pandas as pd
import os
import threading
from datetime import datetime
from PIL import Image, ImageTk

# Ensure directories exist
if not os.path.exists('faces'):
    os.makedirs('faces')
if not os.path.exists('attendance'):
    os.makedirs('attendance')

attendance_list = []
cap = None  # Global camera variable
face_encodings_known = []
face_names_known = []
last_attendance_timestamp = None

# Load known faces
def load_faces():
    global face_encodings_known, face_names_known
    face_encodings_known, face_names_known = [], []
    for file in os.listdir('faces'):
        if file.endswith('.jpg'):
            image = face_recognition.load_image_file(f'faces/{file}')
            encodings = face_recognition.face_encodings(image)
            if encodings:
                face_encodings_known.append(encodings[0])
                face_names_known.append(file.split('.')[0])

# Check if the face is already registered
def is_duplicate_face(face_encoding):
    matches = face_recognition.compare_faces(face_encodings_known, face_encoding,
tolerance=0.45)
    return True in matches

# Register new person
def register_person():
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        messagebox.showerror("Error", "Camera not accessible!")
        return
    while True:
        ret, frame = cap.read()
        cv2.imshow('Register - Press "s" to save', frame)
        if cv2.waitKey(1) & 0xFF == ord('s'):
```

```
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            encodings = face_recognition.face_encodings(rgb_frame)
            if not encodings:
                messagebox.showerror("Error", "No face detected! Try again.")
                break
            if is_duplicate_face(encodings[0]):
                messagebox.showerror("Error", "This face is already registered!")
                break

            name = simpledialog.askstring("Input", "Enter Name and Roll Number (e.g.
John_101):")
            if name:
                cv2.imwrite(f'faces/{name}.jpg', frame)
                messagebox.showinfo("Success", f"{name} registered successfully!")
                load_faces()
            break
    cap.release()
    cv2.destroyAllWindows()

# Delete a person
def delete_person():
    person = simpledialog.askstring("Delete", "Enter Name and Roll Number to delete:")
    if person and os.path.exists(f'faces/{person}.jpg'):
        os.remove(f'faces/{person}.jpg')
        messagebox.showinfo("Deleted", f"{person} deleted successfully!")
        load_faces()
    else:
        messagebox.showerror("Error", "Person not found!")

# Get today's attendance file
def get_attendance_file_path():
    today = datetime.now().strftime("%Y-%m-%d")
    return f'attendance/attendance_{today}.xlsx'

# Update attendance
def mark_attendance(name):
    if name in attendance_list:
        return  # Prevent duplicate attendance
    attendance_list.append(name)
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    file_path = get_attendance_file_path()
    df = pd.DataFrame([[name, timestamp]], columns=['Name', 'Timestamp'])
    if os.path.exists(file_path):
        df_existing = pd.read_excel(file_path)
        df = pd.concat([df_existing, df], ignore_index=True)
    df.to_excel(file_path, index=False)
    messagebox.showinfo("Attendance", f"{name}'s attendance recorded!")
```

```
# Face recognition logic
def recognize_faces(frame):
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    face_locations = face_recognition.face_locations(rgb_frame)
    face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)

    for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
        matches = face_recognition.compare_faces(face_encodings_known, face_encoding,
tolerance=0.45)
        face_distances = face_recognition.face_distance(face_encodings_known, face_encoding)
        best_match = None if len(face_distances) == 0 else
face_names_known[face_distances.argmin()]

        if True in matches:
            name = best_match
            color = (0, 255, 0)
            mark_attendance(name)
        else:
            name = "Unknown"
            color = (0, 0, 255)

        cv2.rectangle(frame, (left, top), (right, bottom), color, 2)
        cv2.putText(frame, name, (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)

    return frame
```

Code Breakdown

1. Importing Required Libraries

```
python
CopyEdit
import cv2
import face_recognition
import tkinter as tk
from tkinter import simpledialog, messagebox, Toplevel
import pandas as pd
import os
import threading
```

```
from datetime import datetime
from PIL import Image, ImageTk
```

- **cv2**: Handles webcam access and face detection.
- **face_recognition**: Used for encoding and matching faces.
- **tkinter**: Provides a GUI for user interaction.
- **pandas**: Stores attendance in an Excel file.
- **os**: Manages file operations.
- **threading**: Ensures smooth execution.
- **datetime**: Handles timestamps.
- **PIL (Pillow)**: Displays images in the Tkinter GUI.

2. Creating Required Directories

```
python
CopyEdit
if not os.path.exists('faces'):
    os.makedirs('faces')
if not os.path.exists('attendance'):
    os.makedirs('attendance')
```

- Ensures that "**faces**" (for storing registered images) and "**attendance**" (for storing Excel files) folders exist.

3. Global Variables

```
python
CopyEdit
attendance_list = []
cap = None
face_encodings_known = []
face_names_known = []
last_attendance_timestamp = None
```

- `attendance_list`: Stores already marked attendance to prevent duplicates.
- `cap`: Stores the camera object.
- `face_encodings_known`: Stores encoded face data.
- `face_names_known`: Stores names of registered persons.
- `last_attendance_timestamp`: Keeps track of the last attendance time.

4. Loading Known Faces

```
python
CopyEdit
```

```
def load_faces():
    global face_encodings_known, face_names_known
    face_encodings_known, face_names_known = [], []
    for file in os.listdir('faces'):
        if file.endswith('.jpg'):
            image =
face_recognition.load_image_file(f'faces/{file}')
            encodings =
face_recognition.face_encodings(image)
            if encodings:

face_encodings_known.append(encodings[0])
                face_names_known.append(file.split('.')
[0])
```

- Loads all registered faces from the `faces` folder.
- Extracts **face encodings** and stores them in `face_encodings_known`.
- Extracts **names** from file names and stores them in `face_names_known`.

5. Checking for Duplicate Faces

```
python
CopyEdit
def is_duplicate_face(face_encoding):
    matches =
face_recognition.compare_faces(face_encodings_known,
face_encoding, tolerance=0.45)
    return True in matches
```

- Compares a new face with existing ones.
- Returns `True` if a match is found, preventing duplicate registration.

6. Registering a New Face

```
python
CopyEdit
def register_person():
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        messagebox.showerror("Error", "Camera not
accessible!")
        return
    while True:
```

```
        ret, frame = cap.read()
        cv2.imshow('Register - Press "s" to save',
frame)
        if cv2.waitKey(1) & 0xFF == ord('s'):
            rgb_frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
            encodings =
face_recognition.face_encodings(rgb_frame)
            if not encodings:
                messagebox.showerror("Error", "No face
detected! Try again.")
                break
            if is_duplicate_face(encodings[0]):
                messagebox.showerror("Error", "This
face is already registered!")
                break

            name = simpledialog.askstring("Input",
"Enter Name and Roll Number (e.g. John_101):")
            if name:
                cv2.imwrite(f'faces/{name}.jpg', frame)
                messagebox.showinfo("Success", f"{name}
registered successfully!")
                load_faces()
            break
    cap.release()
    cv2.destroyAllWindows()
```

- Captures a frame from the camera.
- Detects a face and checks if it's already registered.
- If it's a **new face**, asks for the **Name and Roll Number**.
- Saves the image and updates the database.

7. Deleting a Person

```
python
CopyEdit
def delete_person():
    person = simpledialog.askstring("Delete", "Enter
Name and Roll Number to delete:")
```

```
    if person and os.path.exists(f'faces/
{person}.jpg'):
        os.remove(f'faces/{person}.jpg')
        messagebox.showinfo("Deleted", f"{person}
deleted successfully!")
        load_faces()
    else:
        messagebox.showerror("Error", "Person not
found!")
```

- Asks the user for a name.
- Deletes the corresponding **face image** from the system.

8. Managing Attendance

```
python
CopyEdit
def get_attendance_file_path():
    today = datetime.now().strftime("%Y-%m-%d")
    return f'attendance/attendance_{today}.xlsx'
```

- Generates a daily attendance file.

```
python
CopyEdit
def mark_attendance(name):
    if name in attendance_list:
        return
    attendance_list.append(name)
    timestamp = datetime.now().strftime('%Y-%m-%d %H:
%M:%S')
    file_path = get_attendance_file_path()
    df = pd.DataFrame([[name, timestamp]],
columns=['Name', 'Timestamp'])
    if os.path.exists(file_path):
        df_existing = pd.read_excel(file_path)
        df = pd.concat([df_existing, df],
ignore_index=True)
    df.to_excel(file_path, index=False)
    messagebox.showinfo("Attendance", f"{name}'s
attendance recorded!")
```

- Checks if the person is already marked.
- Saves attendance in an Excel sheet.

9. Real-Time Face Recognition

```
python
CopyEdit
def recognize_faces(frame):
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    face_locations =
face_recognition.face_locations(rgb_frame)
    face_encodings =
face_recognition.face_encodings(rgb_frame,
face_locations)

    for (top, right, bottom, left), face_encoding in
zip(face_locations, face_encodings):
        matches =
face_recognition.compare_faces(face_encodings_known,
face_encoding, tolerance=0.45)
        face_distances =
face_recognition.face_distance(face_encodings_known,
face_encoding)
        best_match = None if len(face_distances) == 0
else face_names_known[face_distances.argmin()]

        if True in matches:
            name = best_match
            color = (0, 255, 0)
            mark_attendance(name)
        else:
            name = "Unknown"
            color = (0, 0, 255)

        cv2.rectangle(frame, (left, top), (right,
bottom), color, 2)
        cv2.putText(frame, name, (left, top - 10),
cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)

    return frame
```

- Detects **multiple faces** at once.
- Matches them with registered faces.
- Draws **green boxes** for known faces and **red boxes** for unknown faces.

10. GUI and Camera Controls

```
python
CopyEdit
def update_frame():
    global cap
    ret, frame = cap.read()
    if ret:
        frame = recognize_faces(frame)
        frame = cv2.resize(frame, (640, 480))
        img =
ImageTk.PhotoImage(image=Image.fromarray(cv2.cvtColor(f
rame, cv2.COLOR_BGR2RGB)))
        canvas.create_image(0, 0, anchor=tk.NW,
image=img)
        canvas.img = img
    canvas.after(10, update_frame)
```

- Continuously updates the **camera feed**.

11. Running the System

```
python
CopyEdit
load_faces()
take_attendance()
root.mainloop()
```

- **Loads registered faces**.
- **Starts real-time attendance**.
- **Opens the GUI**.

Final Features

✔ Registers new faces with name & roll number
✔ Prevents duplicate face registration
✔ Marks attendance for multiple faces at once
✔ Stores records in an Excel file
✔ Allows viewing and deleting registered faces
✔ Real-time face detection with Ui