Project 2: Real-time Face Recognition

1. Introduction

The **Real-time Face Recognition System** is a Python-based application that detects and recognizes faces using **OpenCV** and **Local Binary Patterns Histogram (LBPH)**. The system allows capturing and storing facial data, training a recognition model, and identifying individuals in real-time. The main goal of this project is to provide an efficient and accurate face recognition solution for security, attendance tracking, and identity verification purposes.

2. Features

- **Face Detection**: Uses Haar cascade classifiers to detect faces in real-time.
- **Face Recognition**: Utilizes LBPH for recognizing previously registered faces.
- **Data Persistence**: Stores facial encodings and trained models for future use.
- **Multi-Angle Training**: Captures images from different angles (front, left, right, up) for better recognition accuracy.
- **Threaded Execution**: Improves performance using multithreading.
- **Automatic Model Updating**: Adds new faces dynamically and retrains the model accordingly.
- **Real-time Processing**: Optimized face detection and recognition using OpenCV for fast results.
- **User-Friendly Interface**: Displays real-time recognition results with labels and confidence scores.

3. System Requirements

Hardware

- A computer with a webcam (built-in or external)
- At least 4GB RAM for smooth execution
- Minimum 2-core CPU (Recommended: Intel i5 or higher)

Software

- Python 3.x
- OpenCV (cv2)
- NumPy (numpy)
- Pickle (pickle)

- Threading (concurrent.futures)

4. Project Structure

Real-time Face Recognition/

|-- dataset/ # Stores captured images for each person

| |-- person1/ # Folder for person 1's images

| |-- person2/ # Folder for person 2's images

|-- face_data.pkl # Stores face encodings & names

|-- trained_model.yml # Trained face recognition model

|-- trial.py # Main script for real-time face recognition

5. Constants

- DATA_FILE = 'face_data.pkl' - Path for storing face encodings.
- SAVE_FOLDER = 'dataset' - Directory for storing captured images.
- TARGET_IMAGES = 30 - Number of images per person (reduced for testing).
- MIN_FACE_SIZE = 100 - Minimum face size for detection.
- CAPTURE_INTERVAL = 0.5 - Time gap (seconds) between captures.

6. Class: FaceRecognitionSystem

6.1 Initialization (__init__)

- Loads the Haar cascade classifier for face detection.
- Initializes LBPH recognizer (cv2.face.LBPHFaceRecognizer_create()).
- Loads existing face data and trained model.
- Starts webcam feed (cv2.VideoCapture(0)).
- Uses ThreadPoolExecutor for multithreading.
- Defines different capture angles and instructions for training.

6.2 Face Data Handling

- **_load_existing_data()**:
    - Loads stored face encodings from face_data.pkl.
    - Loads the trained model from trained_model.yml if available.
    - Initializes empty lists if no data is found.

- **_save_data()**:
    - Saves face data (encodings and names) to face_data.pkl.
    - Trains and saves the LBPH model to trained_model.yml.
- **is_model_trained()**:
    - Checks if the model is trained by performing a dummy prediction.
    - Returns True if a trained model exists, else False.

6.3 Face Detection

- **detect_faces(frame)**:
    - Converts the frame to grayscale for better detection.
    - Uses Haar cascades to detect faces in the frame.
    - Returns the coordinates of detected faces.

6.4 Face Recognition

- **_recognize_face(face_gray, frame, x, y)**:
    - Recognizes detected faces using the trained LBPH model.
    - Displays name and confidence score on the frame.
    - If an unknown face is detected, initiates training mode.

6.5 Face Training

- **_start_training_new_face()**:
    - Creates a new folder for a new person inside dataset/.
    - Prompts the user to enter their name.
    - Switches the system to training mode.
- **process_focused_face(frame, face)**:
    - Manages face training by capturing images at defined intervals.
    - Ensures controlled image capture using cooldown timers.
    - Stores images in the respective person's folder.
    - Once target images are collected, the training process is triggered.

6.6 Training the Model

- **_train_model()**:

- Loads all images from dataset/.
- Extracts features and labels.
- Trains the LBPH model with collected data.
- Saves the trained model (trained_model.yml).

7. Execution Flow

1. Load the face recognition system.
2. Start webcam feed.
3. Detect faces in real-time.
4. If a known face is detected:
   - Recognize and display the name with confidence score.
5. If an unknown face is detected:
   - Prompt user for name and start training mode.
   - Capture images from different angles (front, left, right, up).
   - Store images in the dataset.
   - Train the model with new data.
6. Save the updated face data and model.
7. Repeat the process for new faces.

8. Challenges & Solutions

Challenges

- **Lighting Conditions**: Poor lighting affects detection and recognition accuracy.
- **Partial Occlusions**: If a person wears sunglasses or a mask, recognition might fail.
- **Processing Speed**: High-resolution images slow down the system.
- **Angle Variability**: If training images are limited to one angle, recognition accuracy drops.

Solutions

- **Adaptive Histogram Equalization** for better contrast.
- **Multiple Angle Capture** ensures faces are trained from different perspectives.
- **Multi-threading** enhances performance and reduces latency.

- **Dataset Expansion** with more diverse training images improves recognition.

9. Future Enhancements

- Implement **Deep Learning models** like CNNs for improved accuracy.
- Support for **multiple cameras** for enhanced security.
- Integration with **cloud storage** for remote face recognition.
- Implement **real-time notifications** for unrecognized faces.
- Develop a **GUI-based interface** for easier management and visualization.

10. Use Cases

- **Attendance System**: Automates attendance tracking in schools, offices, and conferences.
- **Security & Surveillance**: Identifies intruders in restricted areas.
- **Smart Door Locks**: Allows entry to registered individuals.
- **User Authentication**: Replaces traditional passwords with facial login.

11. Conclusion

This system provides a structured approach to **real-time face recognition**, allowing easy registration and recognition of individuals. By improving the training process and incorporating advanced AI techniques, the system can be enhanced further for practical applications.

This project serves as a foundational framework for future developments in real-time face recognition, and with continued enhancements, it can be adapted for commercial and security applications.