Automated Attendance System Documentation

Chirag S
chiragshivakumar123@gmail.com
Vidyavardhaka College of Engineering

June 27, 2024

Contents

1 Introduction

The Automated Attendance System leverages facial recognition technology to streamline the process of recording attendance in various environments. This documentation provides a comprehensive overview of the system architecture, implementation details, and the technologies used.

2 Methodology

2.1 System Design Overview

The automated attendance system operates on a client-server architecture. The client module captures live video streams, detects faces, and communicates with the server module for face recognition tasks. The server manages face recognition using pre-trained models and updates attendance records in real-time.

2.2 Data Collection and Preprocessing

Face images are collected from authorized individuals stored in the `data/photos` directory. Images undergo preprocessing, including face detection and encoding using the `face_recognition` library, ensuring accurate face recognition.

2.3 Face Recognition Model

The system utilizes a deep learning-based approach implemented with the `face_recognition` library. This library employs convolutional neural networks (CNNs) to extract and encode facial features for recognition tasks. The process involves:

- **Feature Extraction**: Extracting facial features from images using pre-trained CNN models.
- **Face Encoding**: Encoding facial features into numerical representations.
- **Recognition**: Comparing encoded features to identify known faces and determine matches.

3 Implementation

3.1 Software Requirements

The system requires Python for development, utilizing libraries such as OpenCV, face_recognition, NumPy, and pandas for face recognition, image processing, and data management.

3.2 Hardware Setup

Client devices with webcams (720p resolution minimum) and a server environment capable of handling real-time face recognition tasks are recommended.

3.3 Coding Structure

The project directory structure includes src/ for main application files (main.py and face_encodings.pkl) and data/ for attendance records (attendance.csv) and image data (photos/).

3.4 Code Snippets

3.4.1 Encoding Faces

```
def encode_faces(directory=PHOTOS_DIRECTORY, model_file=MODEL_FILE):
    """Encode faces and save to a file."""
    known_face_encodings = []
    known_face_names = []
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            filepath = os.path.join(directory, filename)
            image = face_recognition.load_image_file(filepath)
            encodings = face_recognition.face_encodings(image)
            if encodings:
                face_encoding = encodings[0]
                known_face_encodings.append(face_encoding)
                known_face_names.append(os.path.splitext(filename)[0])

    with open(model_file, 'wb') as f:
```

```python
        pickle.dump((known_face_encodings, known_face_names), f)

    print(f"Done encoding faces and saving profiles to {model_file}")
    return known_face_encodings, known_face_names
```

Listing 1: Encoding Faces

Explanation:

- **Purpose:** This function scans a directory (`PHOTOS_DIRECTORY`) containing images and encodes facial features using the `face_recognition` library.
- **Steps:**
  1. Iterates through each file in the directory, filtering for `.jpg` and `.png` files.
  2. Loads each image file and extracts facial encodings using `face_recognition.face_enc`
  3. Stores the face encodings along with corresponding file names (without extensions) in `known_face_encodings` and `known_face_names` lists.
  4. Serializes these lists into a pickle file (`MODEL_FILE`) for later use.
  5. Prints a confirmation message upon completion.

3.4.2 Loading Known Faces

```python
def load_known_faces(model_file=MODEL_FILE):
    """Load known face encodings and names from a file."""
    if os.path.exists(model_file):
        with open(model_file, 'rb') as f:
            known_face_encodings, known_face_names = pickle.load(f)
        print(f"Loaded face encodings from {model_file}")
    else:
        known_face_encodings, known_face_names = encode_faces()
    return known_face_encodings, known_face_names
```

Listing 2: Loading Known Faces

Explanation:

- **Purpose:** This function loads pre-encoded face data from a pickle file (`MODEL_FILE`) if it exists; otherwise, it invokes the `encode_faces` function to generate and save new encodings.

- **Steps:**
  1. Checks if the pickle file (`MODEL_FILE`) exists.
  2. If the file exists, it loads `known_face_encodings` and `known_face_names` from the pickle file.
  3. If the file doesn't exist, it calls `encode_faces` to generate new encodings and saves them.
  4. Prints a message indicating successful loading or encoding.

3.4.3 Making Attendance Entry

```python
def make_attendance_entry(name):
    """Mark attendance for recognized faces using CSV files."""
    if name != "Unknown":
        now = datetime.now()
        date_string = now.strftime('%d/%b/%Y')
        time_string = now.strftime('%H:%M:%S')

        if not os.path.exists(ATTENDANCE_FILE):
            with open(ATTENDANCE_FILE, 'w', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(['Name', 'Date', 'Time'])
                return

        df = pd.read_csv(ATTENDANCE_FILE)

        if not ((df['Name'] == name) & (df['Date'] == date_string)).any():
            with open(ATTENDANCE_FILE, 'a', newline='') as file:
                writer = csv.writer(file)
                writer.writerow([name, date_string, time_string])
                print(f"Attendance recorded for {name} on {date_string} at {time_string}")
```

Listing 3: Making Attendance Entry

Explanation:

- **Purpose:** This function records attendance for recognized faces by appending their names along with the current date and time to a CSV file (`ATTENDANCE_FILE`).

- **Steps:**

1. Checks if the detected `name` is not "Unknown".
2. Retrieves the current date and time strings formatted as day/month/year and hour:minute:second.
3. If the attendance file (`ATTENDANCE_FILE`) doesn't exist, it creates it and writes the header row (`Name`, `Date`, `Time`).
4. Reads the existing CSV data into a DataFrame (`df`).
5. Appends a new attendance entry only if the combination of `name` and `date_string` doesn't already exist in the DataFrame.
6. Prints a confirmation message upon successful attendance recording.

3.4.4 Initializing Webcam

```python
def initialize_webcam():
    """Initialize webcam capture with error handling."""
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open webcam.")
        exit()
    return cap
```

Listing 4: Initializing Webcam

Explanation:

- **Purpose:** This function initializes webcam capture using OpenCV (`cv2.VideoCapture`) and performs error handling.
- **Steps:**
  1. Initializes the webcam capture device (`cap`) with index `0`, which typically represents the default webcam on the system.
  2. Checks if the webcam device (`cap`) is opened successfully.
  3. If the webcam fails to open, it prints an error message and exits the program.
  4. Returns the initialized webcam capture device (`cap`) for further use in the main program.

3.4.5 Processing Frame for Face Recognition

```python
def process_frame(frame, known_face_encodings, known_face_names):
    """Process a single frame for face recognition and mark attendance."""
    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)
    pil_image = Image.fromarray(rgb_frame)
    draw = ImageDraw.Draw(pil_image)

    for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
        matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
        name = "Unknown"

        face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = known_face_names[best_match_index]

        draw.rectangle(((left, top), (right, bottom)), outline=(0, 255, 255))
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
        cv2.putText(frame, name, (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)

        make_attendance_entry(name)

    return frame
```
Listing 5: Processing Frame for Face Recognition

Explanation:

- **Purpose:** This function processes a single video frame captured from the webcam for face recognition and attendance marking.
- **Steps:**
  1. Converts the BGR format video frame (`frame`) to RGB format using `cv2.cvtColor`.
  2. Detects face locations (`face_locations`) in the RGB frame using `face_recognition.face_locations`.

3. Encodes face features (`face_encodings`) at detected locations using `face_recognition.face_encodings`.
4. Converts the RGB frame to a PIL image (`pil_image`) and initializes drawing operations (`draw`) for visual feedback.
5. Iterates over detected face locations and their encodings, compares them with `known_face_encodings`, and identifies matching faces (`name`).
6. Draws rectangles around detected faces and annotates them with the recognized `name` using OpenCV (`cv2.rectangle` and `cv2.putText`).
7. Calls `make_attendance_entry` to record attendance for recognized faces in real-time.
8. Returns the annotated video frame (`frame`) for display in the main program loop.

4 Results

The automated attendance system successfully implements the following key functionalities:

4.0.1 Encoding Faces

- Encodes facial features from images in the specified directory (`PHOTOS_DIRECTORY`).
- Saves encoded face data to a file (`MODEL_FILE`) using pickle serialization.

4.0.2 Loading Known Faces

- Loads pre-encoded face data from the pickle file (`MODEL_FILE`), if available.
- Invokes face encoding if the file does not exist and saves new encodings.

4.0.3 Making Attendance Entry

- Records attendance for recognized faces by appending their details to a CSV file (`ATTENDANCE_FILE`).
- Ensures no duplicate entries for the same person on the same day.

4.0.4 Initializing Webcam

- Initializes webcam capture using OpenCV (`cv2.VideoCapture`).
- Handles errors if the webcam fails to open.

4.0.5 Processing Frame for Face Recognition

- Detects faces in real-time webcam frames and compares them with known encodings.
- Marks attendance for recognized faces by annotating the video feed and updating the attendance CSV.

5 Conclusion

In conclusion, the developed automated attendance system effectively utilizes image processing and machine learning techniques to achieve accurate face recognition and attendance tracking. By encoding facial features, loading known faces, and processing real-time video frames from a webcam, the system accurately identifies individuals and records their attendance in a CSV file. Future improvements could focus on enhancing recognition accuracy, optimizing performance, and expanding functionality for larger-scale applications.

This system serves as a robust foundation for automated attendance management in various educational and organizational settings, offering efficiency and reliability in face recognition technology. With continued development and refinement, it holds promise for broader adoption in practical scenarios.