Pose Estimation and Pose Matching System

This project aims to estimate human body poses from images, videos, and real-time webcam feeds, and to match the similarity between two poses using keypoint detection techniques. The system leverages pre-trained deep learning models and popular computer vision libraries to identify and label key body parts, draw skeletal structures, and provide pose matching results in the form of a similarity percentage. It provides an accessible, easy-to-use interface designed for users with little to no programming experience.

Table of Contents

1. Introduction

The **Pose Estimation and Matching System** is a computer vision-based project designed to identify and track human body parts in images, videos, and live webcam feeds. Using keypoint detection, the system calculates the positions of key body parts and can compare them with other images to determine pose similarity. The project incorporates a user-friendly web interface to make pose analysis accessible to non-programmers, offering a seamless experience for users across various fields, such as sports analysis, fitness tracking, healthcare, and human-computer interaction.

2. Project Overview

This project integrates pre-trained machine learning models for pose estimation, keypoint detection, and pose matching into a fully functional web-based interface. The core components of the system include:

- **Pose Estimation**: Detects human keypoints, such as joints and facial features, from input media (images, video files, or live webcam feeds).
- **Pose Matching**: Compares the keypoints of two poses and returns a similarity score based on the Euclidean distance between keypoints.
- **Real-Time Processing**: The system supports real-time pose estimation and matching using a webcam feed, ensuring real-time analysis and feedback.
- **User-Friendly Interface**: Designed for non-technical users to easily interact with the system without requiring any programming knowledge. Users can upload images or use the live webcam feature to estimate poses and compare them.

Key Features:

- Robust pose estimation for multiple media types (images, videos, and webcam streams).
- A similarity score for pose comparison between two images.
- Web-based interface for ease of use.
- Real-time performance for live video feeds.

3. System Architecture

The system follows a modular architecture, comprising several interconnected components:

- **Client (User Interface):**
  - **Description:** The front-end component through which users interact with the system.
  - **Functions:** Allows image uploads, video recording, and webcam access.
- **Pose Estimation Module:**
  - **Description:** Processes input to detect keypoints and estimate human poses.
  - **Functions:** Uses pre-trained deep learning models to analyze body parts.
- **Pose Matching Module:**
  - **Description:** Compares keypoints from two poses to determine similarity.
  - **Functions:** Calculates similarity scores based on keypoint distances.
- **Backend (Flask API):**
  - **Description:** Facilitates communication between the user interface and the pose estimation/matching modules.
  - **Functions:** Handles API requests and manages data flow between front-end and back-end.
- **Real-Time Data Processing:**
  - **Description:** Processes live inputs from webcams to perform real-time pose estimation.
  - **Functions:** Captures video frames, performs pose estimation, and updates the user interface.

4. Technical Specifications

- **Libraries Used:**
  - **OpenCV:** For image and video processing, including resizing, cropping, and drawing overlays.
  - **TensorFlow:** To utilize pre-trained deep learning models for pose estimation, providing high-level APIs for model loading and inference.
  - **Flask:** Acts as the web framework for handling HTTP requests, managing sessions, and routing API endpoints.
  - **Numpy:** Facilitates numerical computations and matrix operations essential for pose matching.
  - **Matplotlib:** For visualizing results, such as plotting pose skeletons and keypoints on images.
- **Pre-trained Model:** TensorFlow-based model specifically trained for human pose detection. Capable of identifying 18 key body parts.
- **Web Framework:** Flask, providing an easy-to-use interface for managing user interactions and API communication.
- **Deployment:** Local deployment is supported with Flask, with plans for potential scaling to cloud platforms like AWS or Heroku to handle increased traffic and improve reliability.

5. Model Details

The pose estimation model used in this project is a pre-trained TensorFlow model, specifically designed for detecting keypoints on the human body. The model identifies the following key body parts:

- **Nose**, **Neck**, **Shoulders**, **Elbows**, **Wrists**, **Hips**, **Knees**, **Ankles**, **Eyes**, and **Ears**.

The model takes an image or video frame as input and outputs the coordinates of keypoints. These points are then connected to form a skeleton of the person’s pose.

6. Key Functionalities

6.1 Pose Estimation

Pose estimation is the process of identifying key body parts and their positions. This involves:

- Loading a pre-trained neural network for keypoint detection.
- Processing the input (image, video, or webcam feed) to extract the coordinates of each body part.
- Drawing a skeleton overlay on the original image to visualize the detected pose.

Code Example for Pose Estimation:

```
def pose_estimation(frame):

    frameWidth = frame.shape[1]

    frameHeight = frame.shape[0]

    net.setInput(cv.dnn.blobFromImage(frame, 1.0, (inWidth, inHeight),
(127.5, 127.5, 127.5), swapRB=True, crop=False))

    out = net.forward()

    out = out[:, :19, :, :]


    points = []

    for i in range(len(BODY_PARTS)):

        heatMap = out[0, i, :, :]

        _, conf, _, point = cv.minMaxLoc(heatMap)

        x = (frameWidth * point[0]) / out.shape[3]

        y = (frameHeight * point[1]) / out.shape[2]

        points.append((int(x), int(y)) if conf > thr else None)
    return points
```

6.2 Pose Matching

Pose matching compares the keypoints of two poses and calculates the similarity percentage based on the Euclidean distance between corresponding keypoints. This enables the system to determine how similar two poses are.

Code Example for Pose Matching:

```
def calculate_similarity_percentage(keypoints1, keypoints2,
max_distance=200):

    total_distance = 0

    valid_points = 0

    for kp1, kp2 in zip(keypoints1, keypoints2):

        if kp1 is not None and kp2 is not None:

            total_distance += np.linalg.norm(np.array(kp1) -
np.array(kp2))

            valid_points += 1

    average_distance = total_distance / valid_points if valid_points > 0
else max_distance

    return max(0, 100 * (1 - average_distance / max_distance))
```

7. User Interface

The web interface provides an accessible way for users to interact with the system. Key features include:

- **Image Upload:**
    - **Function:** Allows users to upload images for pose estimation.
    - **Implementation:** HTML form with file input control.

- **Live Camera Feed:**
    - **Function:** Enables real-time pose estimation using the webcam.
    - **Implementation:** JavaScript for accessing webcam feed and sending it to the server.
- **Pose Matching:**
    - **Function:** Compares two uploaded images and displays the similarity score.
    - **Implementation:** Interface for selecting and uploading two images.
- **Visualization:**
    - **Function:** Displays detected pose skeletons on the input images or videos.
    - **Implementation:** CSS and JavaScript for rendering the visual output.

Technologies Used:

- **Frontend:** HTML, CSS, JavaScript for creating and styling the user interface.
- **Backend:** Flask for handling API requests and responses, managing user sessions, and integrating with the pose estimation/matching modules.

8. Installation and Setup

Prerequisites:

- Python 3.8+
- Pip (Python package manager)

Setup:

1. Clone the project repository from GitHub.

```
git clone https://github.com/quanhua92/human-pose-estimation-opencv
```

2. Install the required Libraries

```
pip install -r requirements.txt
```

3. Run the Flask Web server locally

```
python app.py
```

4. Open a browser and go to http://localhost:5000 to interact with the web interface.

9. How to Use the Web Interface

- **Upload an Image**
  1. Click the "Upload Image" button.
  2. Select an image file from your computer.
  3. The system will process the image and display the pose estimation results.
- **Live Camera Feed:**
  1. Click the "Use Webcam" button.
  2. Grant access to your webcam if prompted.
  3. The system will start displaying the live feed with real-time pose estimation.
- **Pose Matching:**
  1. Click the "Upload Image 1" button and select the first image.
  2. Click the "Upload Image 2" button and select the second image.
  3. The system will process both images and display the pose similarity score.

10. Testing and Optimization

Testing:

- **Static Images:** Tested with various image datasets to verify accuracy and robustness.
- **Video Files:** Validated pose estimation across different video formats and resolutions.
- **Webcam Feed:** Assessed real-time performance and accuracy under different lighting conditions and user movements.

Optimization:

- **Model Inference:** Optimized by reducing image resolution and adjusting confidence thresholds to improve speed and accuracy.

- **User Interface:** Enhanced responsiveness and usability through CSS and JavaScript adjustments.

11. Challenges and Solutions

- **Real-Time Pose Estimation**: Achieving real-time performance required optimization in data processing and model inference times. Using appropriate thresholds for keypoint confidence and reducing image resolution helped improve performance.
- **Pose Matching**: Handling poses with missing keypoints posed a challenge. This was solved by applying flexible matching algorithms that ignored missing keypoints while calculating similarity.

12. Future Enhancements

3D Pose Estimation:

- **Goal:** Incorporate 3D pose estimation for more detailed analysis of human poses.
- **Approach:** Explore advanced 3D models and integrate with existing system components.

Multi-Person Detection:

- **Goal:** Extend the system to detect and analyze poses of multiple individuals in a single frame.
- **Approach:** Integrate multi-person pose estimation models and adapt the pose matching module accordingly.

13. Conclusion

The Pose Estimation and Matching System is a powerful tool for analyzing human poses, combining advanced machine learning techniques with a user-friendly web interface. It offers accurate pose detection, real-time processing, and robust pose comparison, making it valuable for a variety of applications. The system is designed with scalability and usability in mind, providing a solid foundation for future enhancements.

14. References

1. OpenCV Documentation: https://docs.opencv.org/
2. TensorFlow Models: https://www.tensorflow.org/
3. Flask Documentation: https://flask.palletsprojects.com/en/2.0.x/