**Yoga Pose Detection and Correction Application**

* **Introduction**

This project focuses on creating a Yoga Pose Detection and Correction application utilizing computer vision and machine learning techniques. The application incorporates **MediaPipe** for pose detection and Tkinter for the graphical user interface (GUI). Additionally, a Flask-based API provides pose analysis through a RESTful endpoint. The primary objective is to classify yoga poses and provide corrective feedback to users in real-time or via uploaded images.

* **Approach**

1. **Key Libraries and Tools**

* **MediaPipe**: Used for robust and accurate pose landmark detection.
* **OpenCV**: For image preprocessing and visualization.
* **NumPy**: To facilitate efficient mathematical computations.
* **Tkinter**: Provides the GUI for user interaction.
* **Flask**: Creates a lightweight API to process pose detection requests.
* **Pillow (PIL)**: Converts OpenCV images for Tkinter display.
* **Threading**: Ensures that the Flask API and GUI run concurrently.

1. **Functionalities Implemented**

**Pose Classification:** Identifies poses such as:

* + Warrior Pose
  + Tree Pose
  + Downward Dog
  + Triangle Pose
  + Mountain Pose
  + Cobra Pose

**Feedback Generation**

* + Provides actionable feedback to help users refine their poses.
  + Feedback is based on landmark angles calculated using trigonometric relationships.

**Image Upload & Processing**

* + Users can upload an image to detect their pose and receive corrections.

**Real-Time Detection**

* + Utilizes the webcam to provide continuous feedback on poses in real time.

**Flask API**

* Accepts an image via POST request and returns:
  + Detected pose
  + Corrective feedback
* **Data Processing and Pose Detection**

**1. Preprocessing**

* **Input Format**: Images are processed as RGB arrays using OpenCV.
* **Pose Detection**: MediaPipe extracts pose landmarks, representing key points (e.g., joints) of the body.

**2. Angle Calculation**

The angle between three points (e.g., shoulder, elbow, wrist) is calculated as:

angle = np.abs(radians \* 180.0 / np.pi)

if angle > 180.0:

angle = 360.0 - angle

Angles determine if the pose aligns with the expected configurations for a specific yoga posture.

**3. Pose Classification**

Rules for pose classification are defined based on specific angular thresholds for key joints. Example:

* **Warrior Pose**:
  + Left arm straight (160–180 degrees).
  + Left leg bent (90–120 degrees).

**4. Feedback Mechanism**

Compares detected angles with the ideal range and generates suggestions for improvement.

* **Model Architecture**

MediaPipe's pose detection pipeline consists of:

1. **Pose Detection**: Identifies the overall body structure and orientation.
2. **Landmark Model**: Provides precise 3D positions of 33 body landmarks.
3. **Pose Estimation**: Maps detected landmarks to predefined body poses.

* **Graphical User Interface**

The GUI, built using Tkinter, offers the following features:

1. **Image Upload**:
   * Users select an image to analyze.
   * Displays the processed image with pose landmarks and feedback.
2. **Real-Time Detection**:
   * Initiates webcam-based detection.
   * Continuous feedback updates on the GUI.
3. **Stop Detection**:
   * Terminates the webcam session safely.

* **Flask API Design**

**Endpoint: /api/detect\_pose**

* **Method**: POST
* **Input**: Image file (binary).
* **Output**:

{

"pose": "Warrior Pose",

"corrections": ["Straighten your left arm.", "Adjust your left leg angle."]

}

* **Error Handling**:
  + Returns appropriate error messages for invalid or missing inputs.
* **Results and Observations**

**1. Accuracy**

* MediaPipe provides high-precision pose landmarks with minimal computational overhead.
* Rule-based classification ensures explainable and reliable pose detection.

**2. Real-Time Performance**

* Webcam-based detection achieves smooth and responsive updates, suitable for live feedback.

**3. User Feedback**

* Actionable suggestions improve the user experience and engagement.
* **Next Steps**

1. **Enhance Pose Library**:
   * Add more yoga poses and refine angle thresholds for existing ones.
2. **Machine Learning Model**:
   * Replace rule-based classification with a supervised learning model for greater accuracy.
3. **Mobile Integration**:
   * Develop a mobile-friendly interface for easier accessibility.
4. **Posture Tracking Over Time**:
   * Implement a history feature to track user progress.
5. **Gamification**:
   * Add scoring mechanisms to encourage consistent practice.

* **Conclusion**

This Yoga Pose Detection and Correction application demonstrates a practical implementation of computer vision for real-time health and wellness. By integrating MediaPipe, Tkinter, and Flask, the project offers a comprehensive solution for pose analysis and feedback. Future enhancements aim to improve usability and expand functionality.