Pose Detection and Classification with MediaPipe

**## Introduction**

This documentation covers the setup, usage, and challenges of implementing pose detection and classification using MediaPipe in Python. The example involves loading images, detecting human poses, drawing landmarks, and classifying the detected pose to determine its accuracy against a reference pose.

**## Setup**

* **Required Libraries**

To begin, you need to install the following Python libraries:

**- OpenCV:** for image processing.

**- MediaPipe**: for pose detection and landmark drawing.

**- NumPy:** for numerical operations.

**- Matplotlib:** for displaying images.

**Use the following command to install the required libraries:**

pip install opencv-python mediapipe numpy matplotlib

**### Importing Libraries**

***Import the necessary libraries at the beginning of your script:***

import cv2

import mediapipe as mp

import numpy as np

import matplotlib.pyplot as plt

**### Initializing MediaPipe Pose**

Initialize the MediaPipe pose detection solution:

mp\_pose = mp.solutions.pose

pose = mp\_pose.Pose(static\_image\_mode=True, min\_detection\_confidence=0.5)

mp\_drawing = mp.solutions.drawing\_utils

**## Usage**

* Loading and Displaying an Image

**Load an image using OpenCV and display it using Matplotlib:**

image\_path = r"C:\path\to\your\image.jpg"

image = cv2.imread(image\_path)

if image is None:

raise ValueError("Image not loaded correctly. Please check the path.")

plt.figure(figsize=[10, 10])

plt.title("Input Image")

plt.axis('off')

plt.imshow(image[:, :, ::-1]) # Convert BGR to RGB for display

plt.show()

**### Detecting Pose**

**Define a function to detect and draw pose landmarks on the image:**

def detectPose(image, pose, display=True):

img\_copy = image.copy()

results = pose.process(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

if results.pose\_landmarks:

mp\_drawing.draw\_landmarks(image=img\_copy, landmark\_list=results.pose\_landmarks, connections=mp\_pose.POSE\_CONNECTIONS)

if display:

fig, ax = plt.subplots(1, 2, figsize=[20, 10])

ax[0].set\_title("Original Image")

ax[0].axis('off')

ax[0].imshow(image[:, :, ::-1])

ax[1].set\_title("Output Image with Pose Landmarks")

ax[1].axis('off')

ax[1].imshow(img\_copy[:, :, ::-1])

plt.show()

return img\_copy, results.pose\_landmarks

else:

print("No pose landmarks detected.")

return image, None

**### Classifying Pose**

***Define a function to classify the detected pose and compare it with a reference pose:***

def classifyPose(landmarks, image, correct\_pose\_landmarks, display=True):

def compare\_poses(landmarks1, landmarks2):

return 0.7 # Placeholder for pose similarity score

pose\_accuracy = compare\_poses(landmarks, correct\_pose\_landmarks)

threshold = 0.8

pose\_name = "Correct Pose" if pose\_accuracy > threshold else "Incorrect Pose"

overlay\_color = (0, 255, 0) if pose\_name == "Correct Pose" else (0, 0, 255)

if display:

overlay = image.copy()

alpha = 0.4

cv2.rectangle(overlay, (0, 0), (image.shape[1], image.shape[0]), overlay\_color, -1)

cv2.addWeighted(overlay, alpha, image, 1 - alpha, 0, image)

plt.figure(figsize=[10, 10])

plt.title(f"Pose Classification: {pose\_name}")

plt.axis('off')

for landmark in landmarks.landmark:

x = int(landmark.x \* image.shape[1])

y = int(landmark.y \* image.shape[0])

cv2.circle(image, (x, y), 5, (255, 0, 0), -1)

plt.imshow(image[:, :, ::-1])

plt.show()

return pose\_name

**### Running the Pose Detection and Classification**

***Load the image, detect the pose, and classify it:***

image\_path = r"C:\path\to\your\image.jpg"

image = cv2.imread(image\_path)

if image is None:

raise ValueError("Image not loaded correctly. Please check the path.")

output\_image, landmarks = detectPose(image, pose, display=False)

correct\_pose\_landmarks = None # Replace with actual landmarks of the correct pose

if landmarks:

pose\_name = classifyPose(landmarks, output\_image, correct\_pose\_landmarks, display=True)

print(f"The detected pose is: {pose\_name}")

***## Challenges***

**### Pose Detection Accuracy**

- \*\*Lighting and Background\*\*: Pose detection accuracy can be affected by poor lighting and cluttered backgrounds. Ensure the image is well-lit and the background is simple.

- \*\*Pose Complexity\*\*: Complex poses with occluded body parts can be challenging for the model to detect accurately.

**### Pose Classification**

- \*\*Reference Pose\*\*: Obtaining accurate reference pose landmarks is crucial for comparison. Manually annotating these poses can be labor-intensive.

- \*\*Similarity Metrics\*\*: Developing robust similarity metrics to compare detected poses with reference poses is non-trivial and requires careful consideration of joint angles and positions.

**### Performance**

- \*\*Processing Speed\*\*: Real-time pose detection and classification can be computationally intensive. Optimization techniques, such as reducing image resolution, can help improve performance.

By following the setup and usage guidelines outlined in this documentation, you can effectively implement pose detection and classification using MediaPipe in Python.