Pose detection in image using Mediapipe library

Aman Preet Gulati — Updated On March 1st, 2022

Advanced Computer Vision Image Image Analysis Libraries Python

This article was published as a part of the Data Science Blogathon.

In this article, we will be doing pose detection using **Mediapipe and OpenCV.** We will go in-depth about all the processes and code for the same later in the article but before **let's understand some real-world use cases of Pose detection.**

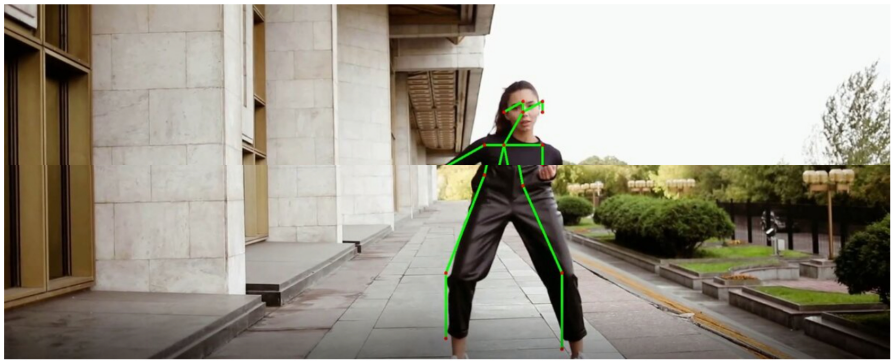


Image Source: Automatic Addison

Application of Pose detection

1. **Fitness application:** This is one of the most modern use cases of pose detection and the fitness industry is boosted by the same.
2. **Camera Surveillance:** As thieves are getting smart day by day it's time for us to make smart cameras with the help of pose detection and detect each moment of them.
3. **Filters:** To make more realistic, interactive, and modern body filters for the applications which are widely being used by all age groups of people.

Note: I will be doing all the coding parts in the Jupyter notebook though one can perform the same in any code editor yet the Jupyter notebook is preferable as it is more interactive.

So let's build our very own pose detection app.

Import the Libraries

Let's import all the libraries according to our requirements.

```
import cv2
import mediapipe as mp
import matplotlib.pyplot as plt
```

Image Source: Mediapipe

Initialize the Pose Detection Model

Initialize the Pose Detection Model

So from here our very first step begins **which is to initialize our pose detection model**, this step is very much common in terms of creating the model as before passing any custom value to the model **we first need to provide some compulsory model params and for the betterment, this case will also help in model testing later in the process.**

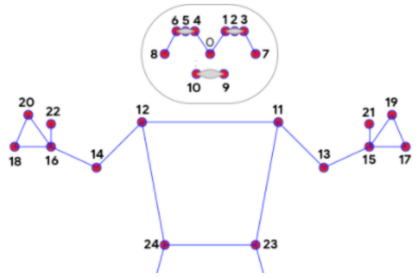
```
# Initialize mediapipe pose class.
mp_pose = mp.solutions.pose

# Setup the Pose function for images - independently for the images standalone processin
pose_image = mp_pose.Pose(static_image_mode=True, min_detection_confidence=0.5)

# Setup the Pose function for videos - for video processing.
pose_video = mp_pose.Pose(static_image_mode=False, min_detection_confidence=0.7,
                          min_tracking_confidence=0.7)

# Initialize mediapipe drawing class - to draw the landmarks points.
mp_drawing = mp.solutions.drawing_utils
```

Code-breakdown

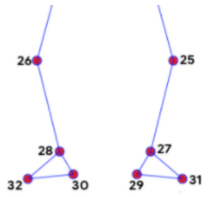
1. Hence, we will first initialize the media pipe pose main class i.e. **solutions. pose with a variable that will hold the instance of that media pipe pose detection model.**
2. Now as we have stored the instance of pose detection class in `mp_pose` variable now we will use this variable to call the Pose method as `mp_pose.Pose()` and store it in `pose_image` In this way, we will set the function for pose estimation. Now let's discuss the parameters in the Pose function.
    1. `static_image_mode`**: It holds the boolean value (True/False), by default it gets initialized with the false** which means that it will treat the given images in the format of videos and ideal processing of each image will not be possible but when we switch it up to True then it will treat the given image in the ideal fashion separate processing will be there i.e. **it will not treat the given image as a video stream and that is our requirement here.**
    2. `min_detection_confidence`**: As the name suggests this argument will set the threshold value of the confidence level and it ranges from [0.0, 1.0]** i.e. minimum confidence level is 0 and maximum confidence level is 1 and by default the value is 0.5, here we are setting the threshold value to 0.5.
3. Previously we have set the function for images now we will do the same for video streaming for that we will use the same `Pose` function and store the instance of the same in `pose_video`. Let's look into its parameters as well.
    1. `static_image_mode`**:** Here we require to stream it for the videos then for that reason **we will set the value to False.**
    2. `min_detection_confidence`**:** Here we have set the **confidence threshold to 0.7**
    3. `min_tracking_confidence`**:** This is the tracking level confidence i.e. it will detect whether the person is detected or not based on the confidence level that we provide so that the problems of high robustness should not occur after the detection. **Note: It will automatically get ignored when `static_image_mode` is set to True.**
4. Now at the last, we will use the `drawing_utils` function to **draw all the landmarks points on the image** or the video (live stream) that our model will detect from the above processing.

Perform Pose Detection

So now, we have overcome all the prerequisites to start with pose detection using Mediapipe hence, **now we will make a pose detection function that will help us to perform the detection in a modular way** and makes the whole pipeline more efficient.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee



10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Image Source: Mediapipe

First, let's understand the functionality of this function in a nutshell then we will break down the code: **This function will perform the pose detection on the person who is closely associated with the image and draw all the landmarks points in the pose which will be detected in the image.**

Arguments:

- **image:** This argument will demand the image in which the person is there to be detected.
- **pose:** The pose function which we created for images and video one of those will be required according to the requirements
- **draw:** This argument will have the boolean value when the value is True which means the function should draw the landmarks points otherwise not.
- **display:** This argument will also expect the boolean value and if it has the True value then it will show both the input image and the resultant image otherwise not.

Output:

- **output_image:** It will return the input image with the landmarks points which were detected in the person's pose.
- **results**: It shows the output on the input image i.e. it is responsible for the visibility of the output on the image.

```
def detectPose(image_pose, pose, draw=False, display=False):

    original_image = image_pose.copy()

    image_in_RGB = cv2.cvtColor(image_pose, cv2.COLOR_BGR2RGB)

    resultant = pose.process(image_in_RGB)

    if resultant.pose_landmarks and draw:

        mp_drawing.draw_landmarks(image=original_image, landmark_list=resultant.pose_lan
                                  connections=mp_pose.POSE_CONNECTIONS,
                                  landmark_drawing_spec=mp_drawing.DrawingSpec(color=(25
                                                                               thickness
                                  connection_drawing_spec=mp_drawing.DrawingSpec(color=(
                                                                               thickness

if display:

        plt.figure(figsize=[22,22])
        plt.subplot(121);plt.imshow(image_pose[:,:,::-1]);plt.title("Input Image");plt.a
        plt.subplot(122);plt.imshow(original_image[:,:,::-1]);plt.title("Pose detected I

else:

    return original_image, results
```

Code-breakdown:

1. First, we will be **creating a copy of the original image so that we don't lose anything from the original image** while preprocessing.
2. Later we will be converting the **Blue Green Red format to Red Green Blue format as in computer vision** it is widely used.
3. Now we will be processing the pose detection on the converted **image format using the `process` function.**
4. It's time for some validation and checks whether the landmarks were detected in the image or not later if the **landmarks got detected then we will draw them on the image using `draw_landmarks` function.**
5. Previously first we did the validations before drawing the landmarks now we will check if the display argument is allowing us **to show the resultant and input image or not.**
6. If the above condition is true then the original, **as well as the resultant image, is displayed.**

7. Otherwise, it will **only return the `output image` and the `results`.**

Now we will test the function `detectPose()` created above to perform pose detection on a sample image and display the results.

```
# Here we will read our image from the specified path to detect the pose
image_path = 'media/sample2.jpg'
output = cv2.imread(image_path)
detectPose(output, pose_image, draw=True, display=True)
```

Output:

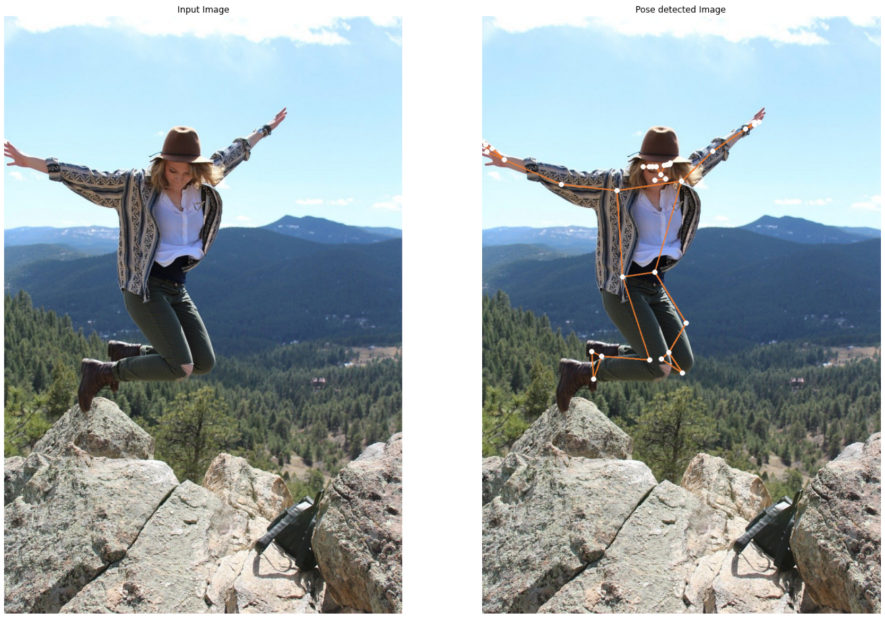


Image source: Author

Conclusion

As we can see that the results are quite pleasing and it is visible that Mediapipe has done quite good work in terms of pose detection **while our `detectPose()` was able to handle the pipeline of the process smoothly and efficiently. You can try with different images to test this function all you have to do is change the path of the image in `image_path`.**

EndNotes

Here's the repo link to this article. I hope you liked my article on Pose detection using Mediapipe. If you have any opinions or questions, then comment below.

Read more articles on our blog about Computer Vision.

About the Author

Greeting to everyone, I'm currently working *in **TCS** and previously, I worked as a Data Science Analyst* in **Zorba Consulting India.** Along with full-time work, I've got an immense interest in the same field, i.e. Data Science, along with its other subsets of Artificial Intelligence such as Computer Vision, Machine Learning, and Deep learning; feel free to collaborate with me on any project on the domains mentioned above (LinkedIn).

The media shown in this article is not owned by Analytics Vidhya and are used at the Author's discretion.

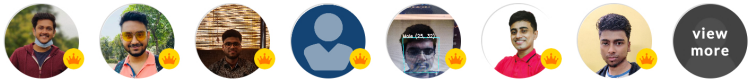
Pose Estimation using OpenCV
Facial Landmark Detection Simplified With Opencv
Detect the Faces in the Image using the Mediapipe Library

blogathon MediaPipe Pose Detection

About the Author

Aman Preet Gulati

Our Top Authors



Download
Analytics Vidhya App for the Latest blog/Article

Previous Post
Handling forms in Flask with Flask-WTForms

Next Post
Gaussian Naive Bayes Algorithm for Credit Risk Modelling

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

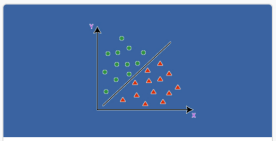
Email*

Website

Notify me of follow-up comments by email. Notify me of new posts by email.

Submit

Top Resources



10 Best AI Image Generator Tools to Use in 2023

avcontentteam - AUG 17, 2023

Understand Random Forest Algorithms With Examples (Updated 2023)

Sruthi E R - JUN 17, 2021

How to Read and Write With CSV Files in Python?

Harika Bonthu - AUG 21, 2021

Guide on Support Vector Machine (SVM) Algorithm

Anshul Saini - OCT 12, 2021