NN&GA Project: A Pose Classifier

Nuh Furkan Erturk - 1231EB

Introduction

MediaPipe is a strong tool developed by Google that offers a suite of solutions. One of those solutions is called MediaPipe Pose Landmarker[1]. In this investigation a pose classification model was prepared and a user interface module was developed for real time interaction with the model. In order to process user data MediaPipe Pose Landmarker tool was used.

The project consists of three distinct faces. All of them together help users capture real life images instantly and classify them on a server. For the user interface a web application developed. The application served on a Flask server. In the flask server a class of MPObject developed for image processing and landmark extraction.

The other facade of the project is the preprocessing and developing the classifier. For this purpose “Physical Exercise Recognition” dataset[2] was used. A variety of processes applied on the raw data obtained and a final version for the production prepared. In the Flask server, the MPObject objects utilised the final processed data.

MediaPipe Pose LandMarker

In this section I aim to describe MediaPipe Pose LandMarker in depth according to its usage in this investigation.

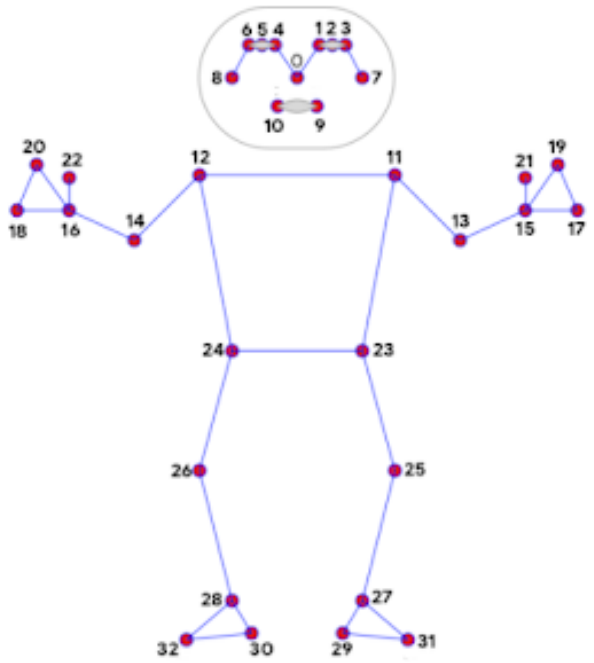
Pose Landmarker Model is developed by Google, readily available for web, android and python environments. In this project python libraries were used.

Pose Landmarker is included in the package “mediapipe”. After downloading mediapipe package user should set configurations and download an additional model amongst:

- Pose Landmarker (Lite)
- Pose Landmarker (Full)
- Pose Landmarker (Heavy)

I have used “Pose Landmarker (Heavy)”. As the model gets heftier the possible frame processing per second drops, however accuracy increases[3]. Since we have no streamed data to process and the processing occurred in the server (whihc in my case my computer) the efficiency was not an issue. So, more accuracy was preferred.

MediaPipe Pose Landmarker returns an object of PoseLandmarkerResult[4]. The object returned contains 33 landmarks regarding the input image. In the “*Figure 1: Illustration of Pose Landmarker Model*”, you can see the output model.



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

Figure 1: Illustration of Pose Landmarker Model

Physical Exercise Recognition Dataset

Physical Exercise Recognition data set is served on Kaggle and openly available. According to their kaggle page the data was used for a closed Kaggle Competition and all the information. There were 1371 entries in the dataset for 10 different classes. In the original data-set landmark coordinates, labels, angles, 3-d distances and xyz-distances were shared. However, in our investigation, I have used only the labels(labels for the entries with the corresponding IDs) and landmark coordinates.

The motivation of using the landmark coordinates was to avoid additional and unnecessary processing of the extracted MediaPipe data.

All the data preprocessed before using to train the MLPClassifier. For preprocessing following steps was followed:

1. Data Normalisation

   All the X, Y and, Z coordinates of the entries are normalised. So that all values lay between 0 and 1. So all 33 entries of X values normalised as a whole and the same process applied for Y and Z coordinates as well. Process applied for the whole process
2. Extract the normalised data

   Normalised Data extracted to a csv file named as “”normalized_landmarks.csv
3. Reset the data-frame

   “Pose_id” column removed from the data to not interfere with the training
4. Train Modal

   A MLPClassifier was trained by using the data-frame created. The modal training phase explained in depth in the next section.

Data Processing

For data processing the “sklearn.neighbors”, and “sklearn.neural_network” packages were used. After the preprocessing explained in the previous chapter, a test model was generated, data was split with seven percent test set and ten percent test set for MLPClassifier and KNNClassifier correspondingly.

For MLPClassifier several different parameters were tested for best results. For example, for the solver “lbfgs” and “adam” were performed around eighty percent accuracy while “sgd” performed around ten percent accuracy.

Max iteration was set as five hundred and kept the same for all the tests, as it was well tested previously during the course activities and homeworks.

For hidden layer sizes I have used the following formula found in a stackexchange blog entry[6]:

$$N_h = \frac{N_S}{\alpha \times (N_i + N_o)}$$

Formula 1: Calculation of the hidden layer size

Where N_h is the Hidden Neurons; N_i is the input neurons, N_h is the output neurons, N_s is the samples in the training data-set and, **α** is a constant coefficient to be determined which can take a value between 2-10. For this investigation I have taken input neurons as one, output neurons as ten and hence tested my classifier with the hidden neurons around 50-100. Eventually I found out that eighty performs well.

Userface

Userface is the web application developed using web technologies and flask. In this section we are going to look for two sides of this application. One at the user end, the other one in the servers.

The user-end of the “userface”, has a single web page with a video element showing user-camera output. A canvas showing the last shot taken. There are two buttons, one of which to start the camera and the other one for shooting pictures. And there is a paragraph element to show the feedback, taking a jinja parameter.

On the server side, most files was kept in the “serverfiles” folder, which includes following files:

- mpprocess.py
- Pose_landmarker_heavy.task
- Related pickled MLPClassifier Modal

Only the serving “app.py” file is kept in the root directory.

“mpprocess.py” has a class to take image arguments and return a feedback with relevant captions.

First thing done in the MPObject call, is to initialise ‘mediapipe' configurations, fetch the image from the localhost, read csv files, and prepare the MLPClassifier.

In the “app.py” file when a http request is received with an image upload, the method “fetchResults” is called. This methods does the following steps in the given order:

- Call “retrieveLandmarks” method
  Which retrieve the mediapipe PoseLandmarkerResult object
- Call “putDataInFrame” method
  A method returns a dataframe from the PoseLandmarkerResult object which is compatible with the MLPClassifier modal I have used. It puts the data in a pandas frame so that I can manipulate and access it with ease.
- Call “normaliseData” method
  A method, normalising the received landmarks
- Call “runMLPC” method
  Runs the classifier and returns a feedback

After running the “fetchResults” method, the feedback is printed on the user screen.

Conclusion

This investigation helped me to understand a variety of ML technologies and the importance of each one of them.

I have tried to make the software as compatible as I can. However, I know that there are still issues related to compatibility. In the future versions those parts should be improved.

The accuracy of the classifier is over ninety percent which is assertable. On the other hand the KNClassifier I have tested in the preprocessing phase performed with over 99 percent accuracy.

Bibliography

1. Goole(2023), MediaPipe, Available at: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker (Accessed 27.05.2023)
2. Kaggle(2023), Physical Exercise Recognition Dataset, Available at: https://www.kaggle.com/datasets/muhannadtuameh/exercise-recognition?resource=download&select=labels.csv (Accessed 03.09.2023)
3. Goole(2023), MediaPipe Pose Landmarker, Available at: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker/index#models (Accessed 27.05.2023)
4. Goole(2023), MediaPipe PoseLandmarkerResult, Available at: https://developers.google.com/mediapipe/api/solutions/python/mp/tasks/vision/PoseLandmarkerResult (Accessed 27.05.2023)
5. StackExchange(2023), How to choose the number of hidden layers and nodes in a feedforward neural network?, Available at: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw (Accessed 03.09.2023)