AI Fitness Assistant

Ioannis Ioannidis - Dimitris Patiniotis Spyropoulos

University of Piraeus
NCSR Demokritos

Keywords— AI Fitness Assistant, Machine Learning, Pose Estimation, SVM, OpenCV, MediaPipe, Classification

Abstract: This project addresses the idea of introducing a virtual fitness assistant. Specifically, we will train a classifier using machine learning methods and data available on YouTube, without publishing any of the original videos or sounds. This classifier will be used in order to build a web-app, which will be keeping the records of a live feed workout.

1. Introduction

Staying at home for long periods of time has - unfortunately - become part of our everyday lifes due to the current situations of pandemic and lockdowns. However, this cannot be a relevant excuse for being unproductive and we need to keep on with our daily routine with respect to staying at home limitation. Working out constitutes an integral and healthy part of such a routine and it is obvious that it is intensively affected by the social isolation. In this project we tried build a tool, which would help both trainers and trainees for hybrid sessions, by reporting the full workout that someone is running at home, just by using the camera of the computer.

In the present report we describe the steps we followed, in order to build this tool. Firstly, in the *Data Description* we provide some information about the data we gathered, the transformations we applied and the final dataset form. After that, in the *Methodology* section we give an intuitive taste of the basic libraries we used and we present the path-planning up to building the final app, while some of the results and performance are reported in the *Results* section.

2. Dataset Description

For the project we gathered optical data from Youtube. Specifically, we used the utility of hasttag (#), that it provides in order to find relevant videos for our topic. We searched for workouts including the exercises we are interested in : pull-ups, squats, and plank. We cropped videos in the part we are interested in, which was a front side, full body observation performing the desired exercise.

However, none of the video information became public as we didn't have the needed licences. We just used them in order to extract the coordinates of specific body parts, while running the workout. This was achieved with the usage of Python's library MediaPipe. More information about how MediaPipe works and what we did are provided in the *Methodology* section.

In that way, we ended up having numerical features about the body parts, when a specific exercise is performed. It is remarkable that the created dataset is scaled, as all coordinates are measured in the same system provided by the MediaPipe.

We used a sample size of 8-10 videos for each exercise, added by the same number of "resting" videos, in order to create one more class, where the trainee doesn't performs any part of the program, but rests. Final dataset consists of 9290 observations and 67 features in a format as shown in Figure 1.

3. Methodology

3.1. Background

3.1.1. OpenCV Library

OpenCV is an open-source library mainly used for computer vision, image processing, and machine learning, which provides great output for real-time data. We can process images and videos so that the implemented algorithm learns to identify objects such as cars, traffic signals, number plates, etc., bodies, faces, or even human handwriting. With the help of other data analysis libraries, OpenCV is capable of processing the images and videos according to one's desire.

The library which we are going to use along with OpenCV-python is MediaPipe.

	LABEL	NOSE	LEFT_EYE_INNER	LEFT_EYE	LEFT_EYE_OUTER	RIGHT_EYE_INNER	RIGHT_EYE	RIGHT_EYE_OUTER	LEFT_EAR	RIGHT
0	Squats	0.504951	0.514962	0.521597	0.527880	0.490489	0.480893	0.472264	0.533444	0.4
1	Squats	0.526823	0.528318	0.531981	0.535665	0.513403	0.505798	0.497616	0.533351	0.4
2	Squats	0.531405	0.533030	0.536469	0.539779	0.518286	0.510728	0.502580	0.533579	0.4
3	Squats	0.527622	0.530922	0.535011	0.538833	0.513843	0.505464	0.497221	0.533540	0.4
4	Squats	0.529673	0.532290	0.536041	0.539568	0.516019	0.507488	0.498053	0.533145	0.4
...	...	...	...	...	...	...	...	...	...	
9285	Plank	0.221354	0.207233	0.211548	0.215775	0.195234	0.191813	0.188909	0.227866	0.1
9286	Plank	0.221485	0.207246	0.211566	0.215792	0.195373	0.191997	0.189132	0.227863	0.1
9287	Plank	0.224056	0.209296	0.213489	0.217575	0.197375	0.193857	0.190875	0.229311	0.1
9288	Plank	0.224449	0.210191	0.214352	0.218385	0.198087	0.194433	0.191319	0.229955	0.1
9289	Plank	0.224714	0.210370	0.214518	0.218552	0.198340	0.194704	0.191587	0.230111	0.1

9290 rows × 67 columns

Fig. 1: Final Dataset

3.1.2. MediaPipe Library

Mediapipe is a framework mainly used for building multimodal audio, video, or any time series data. With the help of the MediaPipe framework, an impressive ML pipeline can be built for instance of inference models like TensorFlow, TFLite, and also for media processing functions.

MediaPipe Pose is a framework for high-fidelity body pose tracking, which takes input from RGB video frames and infers 33 3D landmarks on the whole human. Current state-of-the-art approach methods rely primarily on powerful desktop environments for inferencing, whereas this method outperforms other methods and achieves very good results in real-time.

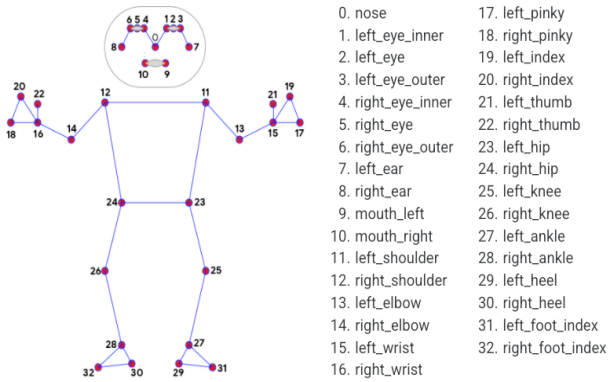


Fig. 2: Full Body Landmarks

3.1.3. Support Vector Machines

Support vector machine (SVM) is a method for the classification of both linear and nonlinear data. If the data is linearly separable, the SVM searches for the linear optimal separating hyperplane (the linear kernel), which is a decision boundary that separates data of one class from another. Mathematically, a separating hyperplane can be written as $W \cdot X + b = 0$, where W is a weight vector and $W = w_1, w_2, ..., w_n$, X is a training tuple and b is a scalar. In order to optimize the hyperplane, the problem essentially transforms to the minimization of $|W|$, which is eventually computed as $\sum_{i=1}^{n} a_i y_i x_i$, where a_i are numeric parameters and y_i are labels based on support vectors X_i. That is: if $y_i = 1$, then $\sum_{i=1}^{n} w_i x_i >= 1$; if$y_i = -1$, then $\sum_{i=1}^{n} w_i x_i >= -1$.

If the data is linearly inseparable, the SVM uses nonlinear mapping to transform the data into a higher dimension. It then solve the problem by finding a linear hyperplane. Functions to perform such transformations are called kernel functions. The kernel function selected for this example is the Gaussian Radial Basis Function (RBF): $K(X_i, X_j) = e^{-\gamma ||X_i - X_j||^2/2}$, where X_i are support vectors, X_j are testing tuples, and γ is a free parameter. Figure 3 shows a classification example of SVM based on the linear kernel and the RBF kernel.

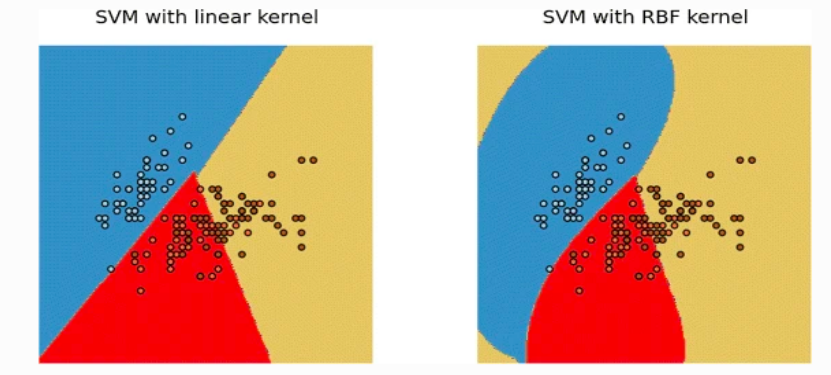
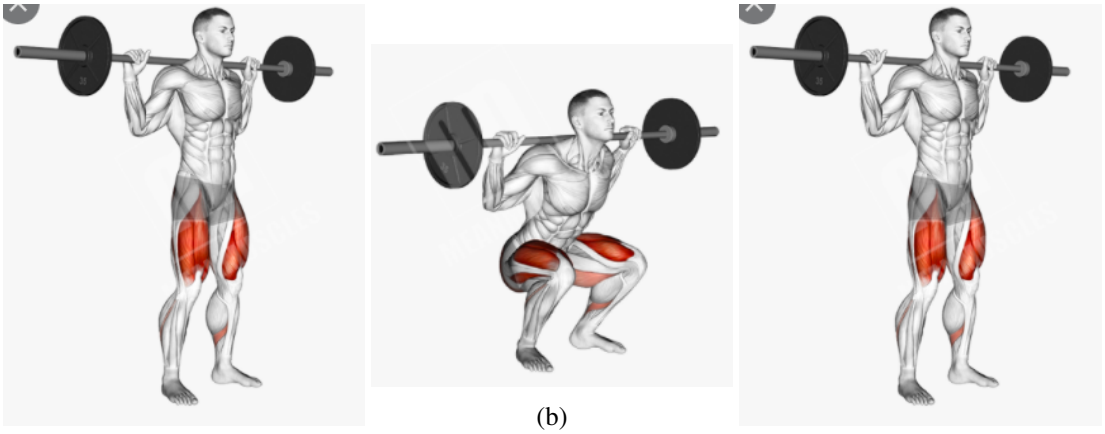


Fig. 3: Classification Example of SVM

3.2. Workflow

Our goal is to detect a person from a front side view and identify each one of the 33 joints that MediaPipe provides. Using the position of those joints, we will train a Support Vector Machine model to classify which exercise is performed. In the present project we will use just three exercises (classes) : pull-ups, squats and plank. In order to get the right features, camera alignment is very important. We need to make sure that the camera looks at the proper front view.

After the model determines the exercise's class based on the samples in the training set, it is now possible to count the repetitions performed. For pull-ups and squats we will be using the property of periodicity. Specifically, it is about exercises, where the trainee passes through the same stages at each repetition. Considering this, we set angle thresholds (for pull-ups we use the (shoulder,elbow,wrist) angle and for squats the (hip,knee,ankle) one) for determining the states of "UP" and "DOWN". For example, we define a rep of squats is a DOWN - UP - DOWN stages alteration, as shown in Figure 4. For the plank exercise, we just count the duration of the unbroken



(a) (b) (c)

Fig. 4: States: DOWN (a) - UP (b) - DOWN (c)

performance by using the python's *time* library.

The total project's workflow is presented in the flowchart in Figure 5.

4. Results

After many fails, we finally got a classifier and implementation working well. We used this model to deploy a web-app, which counts the repetitions of any video displaying pull-ups, squats or plank, achieving good performance. Our app has 3 modules (Fig.6):

- *Demo*
- *Your Video*

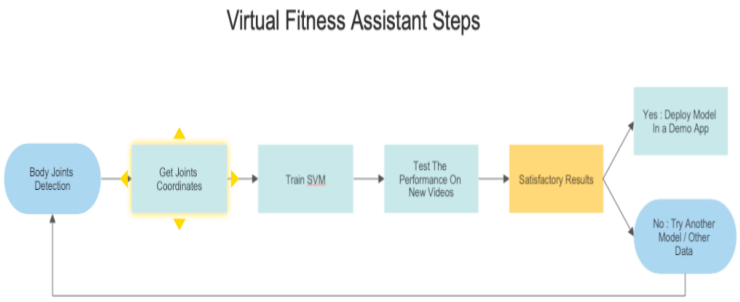


Fig. 5: Workflow

- *About*

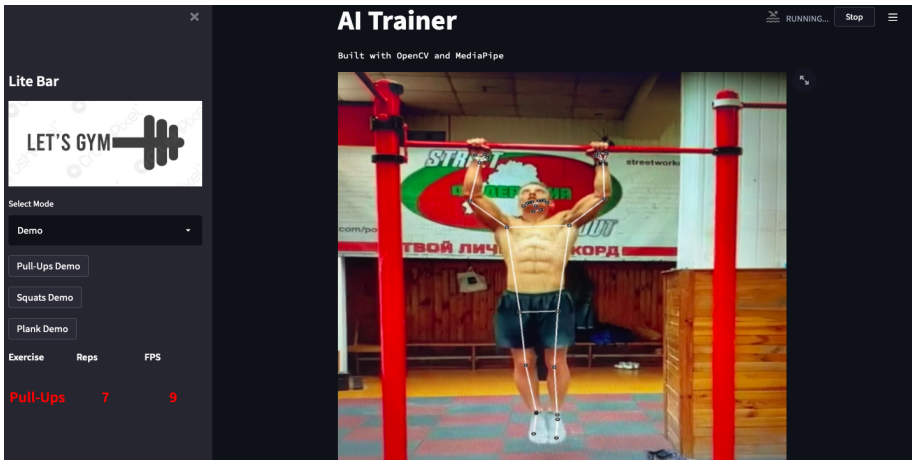


Fig. 6: Web App Environment

Beginning with the *Demo* mode, we created such a feature so that a user can see how the app works, without having to upload any videos or use webcam. In this mode user has the opportunity to select between displaying a pull-up, squat or plank video (Fig.7).

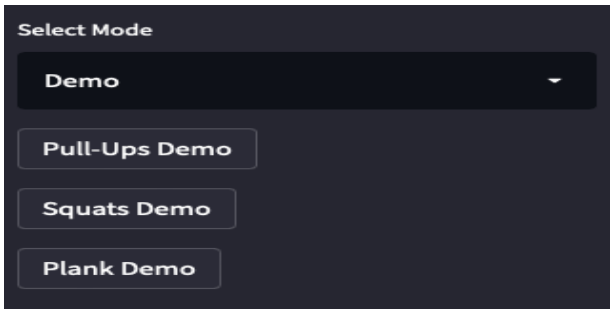


Fig. 7: Demo Mode

Following *Your Video* mode is the main feature, where the user can test the app himself/herself, either by uploading his/her video or by using webcam. Results are previewed in real time, while video is running. (Fig. 8).

Finally, in the *About* mode user can see the purpose we ran this project and some proposed future work.

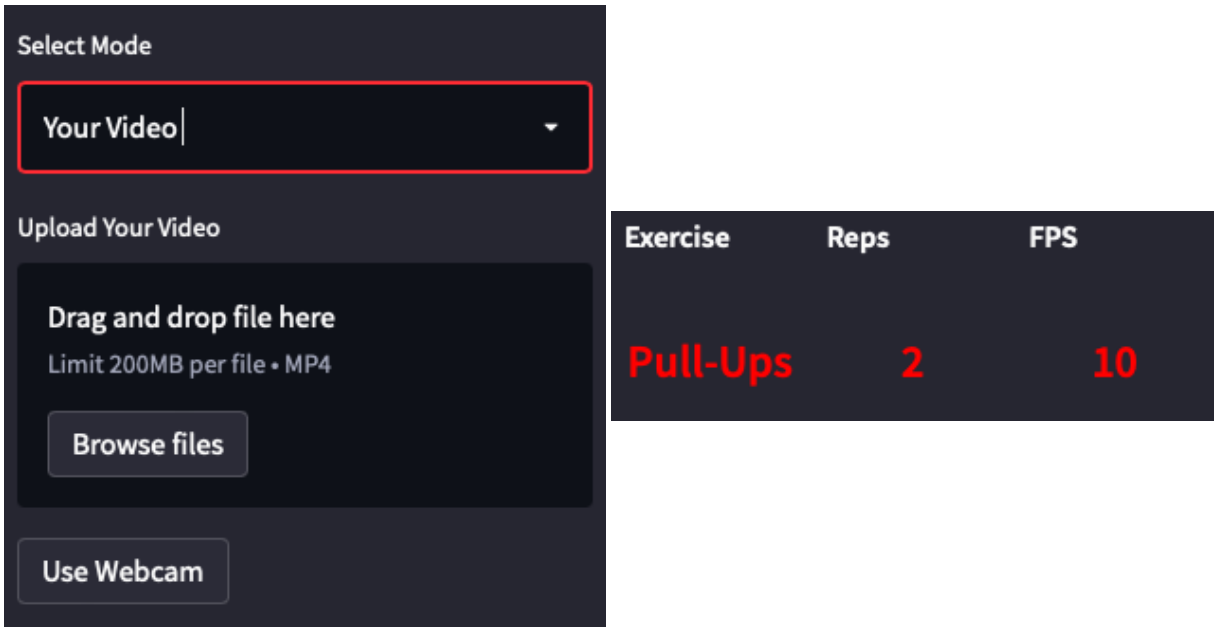


Fig. 8: Your Video Mode