Using Machine Learning to classify various types of human exercises

Rares Mihai Iordache, Alexandru-Iulius Jerpelea

December 2023

Abstract

Artificial intelligence technologies have made their way into various segments of our lives, including fitness and sports. In this paper, we aim to explore the limits of neural networks in regard to human activity. Namely, our scope is to classify different types of exercises by tracking acceleration and angular speed on both hands and legs. To do this, we have also designed our own specified bracelets.

1 Introduction

In the last years, the tech world has seen massive breakthroughs, many of which are based on Artificial intelligence and NN (Neural Networks). One of the areas that has experienced a rise in the use of these algorithms is Motion Recognition, which tries to analyze and label different types of human movements.

This project is part of the Motion Recognition sector and aims to distinguish between six types of movements that could be done during a fitness session: walking, running, jumping jacks, normal stand-still jumping, rotating the arms forward, and rotating the arms backward.

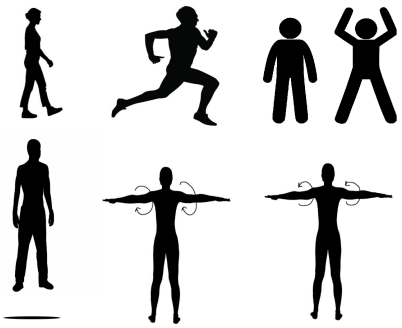

Figure 1: Walking, Running, Jumping Jacks, Stand still jumping, Rotating arms forward and backward - in this order

To do this, we built four bracelets, each for a different body part (left wrist, right wrist, left ankle, right ankle). These bracelets each incorporate a gyroscope and an accelerometer, measuring these data on three axes: X, Y, and Z. We collected data from 23 different subjects, of various ages and physical conditions. Every exercise took around 2-3 minutes, which means that it took us approximately 20 minutes to collect data from each individual.

2 Hardware for acquiring data

As said, we faced the problem of the hardware required for data collection. We wanted full control over this process so we designed our own devices: four lightweight bracelets that are easily attachable/detachable from both the wrists and ankles. Except for the straps, the devices are indistinguishable one from another.These bracelets have the purpose of collecting data about acceleration and orientation (angular velocity).

The main driver of the whole gadget is, of course, the microcontroller. We chose ESP32 for its reliance and speed (other options would have consisted of ESP8266, but we opted for the former). Fixed on a custom-cut perfboard, it is connected to the other components through wires. All wires were soldered to the other components.

Regarding the other components, we used the 6-axis MPU6500 (3-axis for the accelerometer and 3-axis for the gyroscope). Moreover, the device is rechargeable so we used a 3.7V, 400mAh lithium accumulator and a charging module.

Finally, we had custom switches to navigate through functionalties of the bracelet and led lights that indicated the status of the device.

For the actual bracelets, we built them using laser-cut plexiglass. We were also careful to add useful features to the casing. A small hole has been cut in the material to facilitate a fast connection with the computer via a micro USB cable, as well as foam on top of the upper cover, which helps prevent the sensor from moving. The strap was made out of velcro.

This whole setup is indeed DIY but proved itself reliable and accurate enough for our scope, as we will show within the software sections.

Figure 2: Bracelet on left wrist

3 Software for acquiring data

The bracelets use the popular ESP32 microcontroller because of its wide variety of software features. To send data from the devices to our computer, we connected all 4 ESPs and a laptop to a custom network and sent the raw data using the UDP protocol to the computer.

The process was quite simple: from the moment a bracelet is turned on, it starts sending data to the computer through what we will call by now a sample. Such a data sample from an ESP consists of a 6-value tuple (acceleration on the X-Y-Z axis and angular velocity from X-Y-Z). This tuple is accompanied by a timestamp (millisecond passed since the start of the respective day) from a global clock and the id of the bracelet (from 1 to 4).

The timestamp data later helps us associate samples from four bracelets one to another. Namely, we wanted to associate a sample from bracelet "x" with the other 3 samples (from different samples) such that their timestamps are as close as possible. Then we could say that this whole bigger sample consists of accelerometer and gyroscope data from all four devices, on all 6 axes, at a given point in time. This helped a lot in synchronization and building coherent data to feed to the Neural Network, and the "timestamp" method was the best one, given our circumstances.

We tried our best to optimize the process and set a custom difference between sent samples of one specific bracelet. For most networks, this delay was set to 4ms to avoid the risk of a network bottleneck.

We mentioned how the bracelets send data continuously. To ensure we only capture relevant data (that is to say, when a subject is actually doing the desired physical activity), we created software that marks the time intervals in which an exercise is done. We would manually start/stop these intervals and label them

accordingly (participant id, exercise id).

All the codes can be found at:

4 Software for organizing data

We then had to organize all of this data into even bigger "samples" (together with exercise labels), that would later on be given to the Neural Network.

First of all, we organize (as described earlier), samples in groups of 4 (from different bracelets), all around the same timestamp. Then, we sort these groups by the timestamp. To form one training data input, we organize sets of 500 consecutive samples.

Notice how we do this separately for each "interval" corresponding to a specific exercise and individual. Moreover, we rarely also encountered sudden bigger pauses between the samples of one bracelet (probably due to network fluctuations). That's why our software eliminates data from this gap interval and also from other bracelets. In this manner, the data stays coherent and not faulty.

So, one data sample now consists of 6 (acceleration and angular velocity values) * 4 (no. of bracelets) * 500 (chosen bucket size) = 12000 values. Add to this the label (from 1 to 6, corresponding to the exercise), and we get a training example of 12001.

We did this sampling with various overlapping fractions of the 500 chosen samples bucket (this is called stride). We used both 0.2 and 0.5, but stuck to 0.2 because of similar results, but a bigger data volume.

We split our data set into a training data set and a validation data set (around 0.2 of data), using different methods, which are described in the "Results" section.

5 The data acquiring process

Let's now address the procedure for data extraction, with regards to the human element and other technicalities.

The set of 23 subjects was composed of friends, parents, but mainly volunteering students from the Bucharest Polytechnic. As such, we have people of varying physical conditions. From young to old, from unathletic to athletic. We thus hoped to eliminate the bias of our dataset. What is to be said, though, is that most of the subjects are male (22 out of 23). In this regard, our model is probably biased.

We took subjects one by one and attached our equipment to them. We would then time them when doing an exercise for around 2-3 minutes. In between we had breaks to ensure their well-being. Sometimes, our subject required brakes in the middle of an exercise. We could easily ensure that, as we could just stop our registering intervals in our software (discussed eariler). We thus hoped to give our volunteers the most comfortable conditions possible.

All throughout the exercises, we made sure that the participants respected the set conditions for each exercise. Those are:

1. For walking: medium pace, natural hand movement,
2. For running: fast pace, but not sprinting speed,
3. For jumping jacks: standard form, with hands meeting above the head,
4. For stand still jumping: high frequency, low altitude jumping (around 5 - 10 cm from the floor),
5. For rotating arms forward: medium pace, full rotations,
6. For rotating arms backward: same, but backwards.

Notice that we did not require exact speeds and procedures for the movements. In this manner, we can more easily classify variations of the same movements, removing eventual biases.

6 Introduction to NN (Neural Networks)

During the experiment we mainly used fully connected layers. We also did try to use convolutional layers.

A fully connected layer is a layer that connects every node from one layer to all nodes of the next layer, where each node is characterized by a vector of weights and biases, as well as activation functions. A convolutional layer is a partially connected layer, that has an edge only from some of the nodes in the past layer to only some of the nodes in the current layer, usually through kernels. Using these types of layers helps create a complex derivable formula that takes as input the data we collected and organized and outputs the probability of the exercise to be any of the 6 possible physical exercises. We also have a cost function that is preferably convex, so it has a global minimum.

Now, using gradient descent and other optimization algorithms during each iteration (epoch) of the algorithm, we optimize variable values in the nodes to give an ever-increasing in accuracy result. This has the possibility of failing in several ways:

1. The input may be too complex for the Neural Network. In this case, we can simplify it by manually extracting a set of features, training the network for longer, or just expanding the network's size.
2. The network might overfit. This means that the network learns how to classify the training data, but performs poorly on completely new data, implying a certain bias to what it has already seen. We can fix this by using dropout layers or data augmentation.

During this project, we have encountered both problems. We will further present exact results and accuracies at certain steps of our project, and how we managed to improve them.

7 First try: fully connected layers

Our first approach to designing a Neural Network took as a training example the 12000 variables set described earlier and was expected to output a label ranging from 1 to 6. We had to first normalize the input using a min-max scaler for every single variable, across all the data.

The architecture of this simple feed-forward model is as follows:

Type of Layer	**Details**
Input Layer	12000
Dense Layer	16384
Dense Layer	8192
Dense Layer	4096
Dense Layer	1024
Dense Layer	512
Dropout Layer	0.4 dropout prob.
Dense Layer	256
Dense Layer	24
Output Layer	6

Table 1: Feed-forward, basic network - ReLu activation for all layer, except last which uses Softmax, Adam optimizer, 10 epochs, 32 minibatch size, crossentropy loss function

Besides this model, we tried some other variations, but they all led to similar unsuccessful values of accuracy on the validation data (around 0.18). More details regarding accuracies and how the validation samples were extracted can be found in the "Results" section.

The approximately 18 percent is just a bit better than 1 / 6, implying that the model behaves almost randomly, and it's unable to grasp insights from the data.

8 Second try: convolutional NNs

For a training example, if we are to pick a specific axis out of the 6 and also one of the bracelets, we would get an array of 500 numbers, that really just represent a signal. At a very abstract level, signals are 1D images, so we tried to apply convolutional neural networks on them. We had to reorganize the input shape, but all other hyperparameters(choice of normalization method, optimizer, loss function, activation functions, etc.) remained the same.

So for the input, we have a channel(layer) for each bracelet. Every layer has 6 rows (for the 6 axis), and there are 500 values per row. We basically stacked 4 * 6 signals in a volume that we considered best for the given circumstances - obtaining a 4 (depth) * 6 (rows) * 500 (columns) volume.

As for the model arhitecture, we have the following:

Layer	Details
Conv.	128 filters, (1, 100) kernel, stride = 1
Conv.	128 filters, (1, 100) kernel, stride = 1
Conv.	64 filters, (1, 50) kernel, stride = 1
Conv.	64 filters, (1, 50) kernel, stride = 1
Flattening	-
Dropout	0.2 dropout prob.
Dense	512 nodes
Dropout	0.1 dropout prob.
Dense	512 nodes
Output	6 nodes

Table 2: Convolutional Neural Network - ReLu activation for all layers, except last which uses Softmax, Adam optimizer, 10 epochs, 32 minibatch size, crossentropy loss function

We do 1D convolutions, that's why all kernels have 1 row. This model has, unfortunately, produced similar results to our first attempt, with accuracies on the validation data set of around 0.18 (more details in "Results").

9 Third try: feature extraction

At this point, it was clear to us that these models could not, by themselves, figure out any relevant features in the data. Thus, we gathered a set of 8 mathematical features that we extracted from each signal (4 bracelets * 6 axes = 24 signals for a training example). This means that from 12000 variables, we create a new training example with just 24 (signals) * 8 (features per signal) = 192 values. We feed this new data in a simple feed-forward network, with the same hyperparameters mentioned earlier. The results had dramatically improved, but let us first introduce each feature we have extracted from our signal.

We denotate each signal as a sequence $x = (x_k)_{1 \leq k \leq B}$, where B is the number of samples extracted from the signal, in our case 500. Thus, we introduce:

1. Mean Absolute Value (MAV): $MAV(x) = \frac{1}{B} \sum_{i=1}^{B} |x_i|$
2. Zero Crossing Rate (ZCR) - frequency at which the signals passes thorugh zero: $ZCR(x) = |\{x_k \mid 1 \leq k \leq B-1 \; and \; x_k \cdot x_{k+1} \leq 0\}|$
3. Waveform Length (WL) - the total variation seminorm: $WL(x) = \sum_{k=2}^{B} |x_k - x_{k-1}|$
4. Slope Sign Changes (SSC) - measures the frequency at which the sign of the slope changes. $SSC(x) = |\{k \mid 1 \leq k \leq B-1 \; and \; x_{k-1} \cdot x_k \cdot x_{k+1} > 0\}|$
5. Root Mean Square (RMS) - related to the quadratic mean: $RMS(x) = \sqrt{\frac{1}{B} \sum_{k=1}^{B} x_k^2}$

6. Hjorth parameters - this is a set of 3 parameters, but we picked the one we considered to be most relevant: $\sigma^2(x) = \frac{1}{B}\sum_{k=1}^{B}(x_k - \mu(x))^2$, where $\mu(x)$ is the mean value of the signal.

7. Skewness - measures overall asymmetry of probability distribution: $Skew(x) = \frac{1}{B}\sum_{k=1}^{B}(\frac{x_k - \mu(x)}{\sigma(x)})^3$

8. Integrated Square-root EMG(ISEMG) - sum of the fullt-rectified signal: $ISEMG(x) = \sum_{k=1}^{B}\sqrt{|x_k|}$

As for the architecture of the new network processing these values:

Layer	**Details**
Input Layer	192 values
Dense Layer	1024 nodes
Dense Layer	24 nodes
Output Layer	6 nodes

Table 3: Neural Network training for precalculated features - ReLu activation for all layer, except last which uses Softmax, Adam optimizer, 10 epochs, 32 minibatch size, crossentropy loss function

This very simple model has been able to obtain validation accuracies of over 0.95 ("Results" section). We were very skeptical of these results, so we plotted out 10000 random samples from our training data. We picked two random dimensions out of the 192, and thus created a 2D point for each selected sample, giving it a color according to what physical exercise it represents. The following plots correspond to signals from bracelet 1, measuring values on the x-axis.

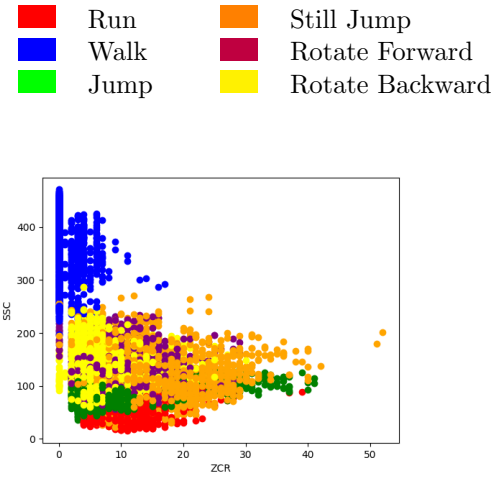

Figure 3: Plotting according to ZCR and SCC

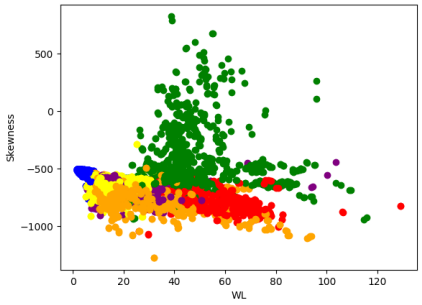

Figure 4: Plotting according to WL and Skew

In this 2D representation, it is fairly easy to (at least partially) outline groups corresponding to each category. In 192 dimensions (the actual input of the neural network), the distinctions become even clearer, thus the high accuracy. Feature extraction does in fact bring a lot of opportunities.

10 Results

Validation accuracies			
NN type	random samples	separate set of people	proportional for each person
Feed-forward, raw data	0.1812	0.1026	0.1420
Convolutional NN	0.1850	0.1102	0.1617
Feature extraction	1.0000	0.9764	0.9445

Table 4: Validation accuracies expressed as fractions (not percentage). For each method of obtaining the validation set, we used $p = 0.2$ out of the total data.

The first type of validation is done on "random" samples. That means that training examples from all subjects are grouped together without consideration of participant id and p percent are selected for the validation set.

The second type of validation takes p-percent of the individuals and uses them as the validation set. This means that the network can not possibly learn any features from these participants, and it better portrays the robustness of the model.

Thirdly, we can select p-percent samples from each individual separately to add to our validation data. This is theoretically more similar to the first method of gathering validation data.

The only successful results can be observed in the NN that is taking already extracted features as input. Its validation accuracy reaches a maximum of 1.0

on the first method used, but the second method demonstrates how that may be the result of the network inherently learning each individual it has been trained on. However, it does behave decently in other circumstances, being able to classify one never seen before individual physical exercise with a 0.97 accuracy.

11 Conclusion

This paper tried to stretch the boundaries of NNs in analyzing human movement. Patterns can be easily identified by these types of models when we already extracted some relevant mathematical features. Human movement is just a signal that can be described by a bunch of characteristics. However, our network did not succeed in classifying movements, based solely on the signals data, as we would have preferred. The fact that we had to manually decide and extract a set of features can be unproductive in other circumstances. However, this could also be the authors' fault. Ranging from noisy data to low computational capabilities, we acknowledge that these could represent a problem in our setup. To conclude, motion recognition is a field waiting to be further explored, and with this paper, we hope to have contributed a bit to its development, but also inspire others to try new ideas and tackle more complex problems.

12 Code and database

Source code for both the hardware and the AI: Github
Our database, alongside documentation: Database

13 Aknowledgments

This project would have not been possible without the help of The Polytechnic University of Bucharest, namely PhD. Ana Neacsu, student Mihai Sumanaru, and master's student Alexandru Guzu which have guided us all throughout. From the required hardware and infrastructure to design the bracelets, helping find volunteers to help create a relevant data set - to the invaluable education, much needed for us to learn about machine learning and actually apply it to a real-life research project such as this one, they were of incredible help to us.

References