




Exermote: Building a fitness tracker with Convolutional LSTM Neural Networks

📅 3. August 2017 (<http://lausbert.com/2017/08/03/exermote/>) 👤 Lausbert
(<http://lausbert.com/author/lausbert/>)

tl;wr: Exermote is a fitness app prototype, which is capable to detect Burpees, Squats and Situps and to count repetitions. Exercise recognition is done with Convolutional LSTM Neural Networks.

It will take some time until .gif is loaded. Have a look on youtube for the raw video (https://www.youtube.com/watch?v=ieolnbYI_TA&feature=youtu.be) with sound. 

The project is divided into following steps:

Gathering Data (<https://github.com/Lausbert/Exermote/tree/master/ExermoteGatheringData>)

Preprocessing And Training

(<https://github.com/Lausbert/Exermote/tree/master/ExermotePreprocessingAndTraining>)

Inference (<https://github.com/Lausbert/Exermote/tree/master/ExermoteInference>)

Checkout links for source files.

Gathering Data

Since the later learning procedure is supervised, labeled data is needed.

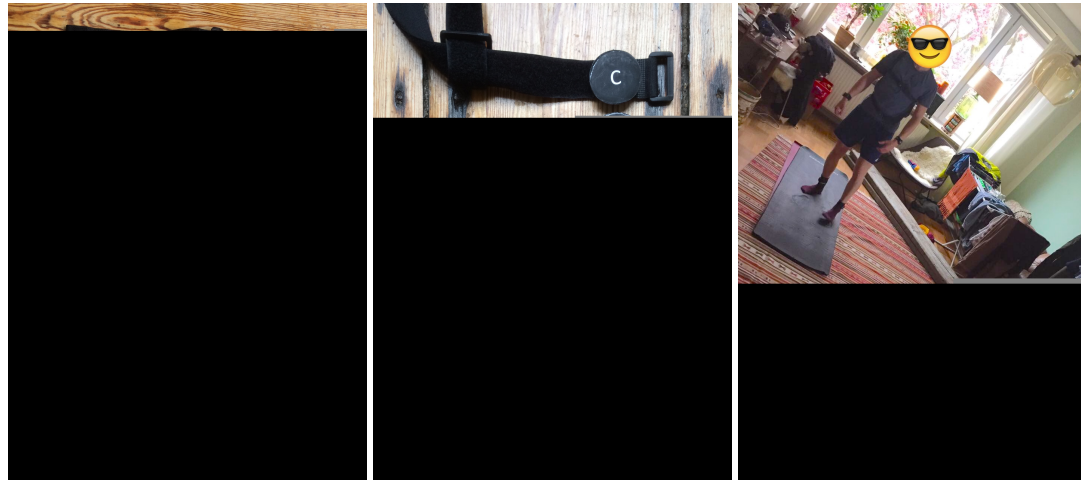
Setting

To record training data of different individuals I used 2 different types of devices:

- Iphone on right upper arm: 12 data features (3x gravity, 3x acceleration, 3x euler angle, 3x rotation rate)
- 6x Estimote Nearables (<https://estimote.com>) on chest, belly, hands and feet: each 4 data features (3x acceleration, 1x rssi)

So there are 36 data features in total. The Nearables were reshaped by using Stewalin, a muffin form and some Velcro cable ties 😊

needed utensils (left), reshaped Nearables (mid), remotely starting recording procedure via firebase (right)



Recording

Recording frequency was 10 Hz for the reason that Nearable send frequency is limited to this value on hardware side. Since the official SDK only allows to get Nearable acceleration data once per second, I had to access and decode advertisement data directly via `CBCentralManager`. Many thanks to reelyactive (<https://github.com/reelyactive/advlib>) for inspiration.

Before recording was started a 5 minute training consisting of “burpees”, “squats”, “situps”, “set breaks” and “breaks” had been generated randomly.

time	exercise type	exercise sub type	36 data feature columns ...
0.1 s	set break	set break	
0.2 s	set break	set break	
0.3 s	set break	set break	
0.4 s	set break	set break	
0.5 s	set break	set break	
0.6 s	burpee	first half	
0.7 s	burpee	first half	

time	exercise type	exercise sub type	36 data feature columns ...
0.8 s	

To ensure that exercising individuals trained accordingly to the pre-generated training and therefore labels matched perfectly to recorded movement data, the app gave spoken hints which exercise will follow. Additionally there was a generated whistle, whose pitch decreased during first half and increased during second half of an exercise repetition.

Raw data

(<https://github.com/Lausbert/Exermote/tree/master/ExermotePreprocessingAndTraining/RawData>) contained 3 hours (=108000 data points) of 6 individuals and was saved to my iCloud drive, when recording was finished.

Preprocessing and Training

After collecting labeled data, a model needs to be trained!

Preprocessing

There were a few preprocessing steps I made. Some of them are rooted in insights I had, when I was already training models:

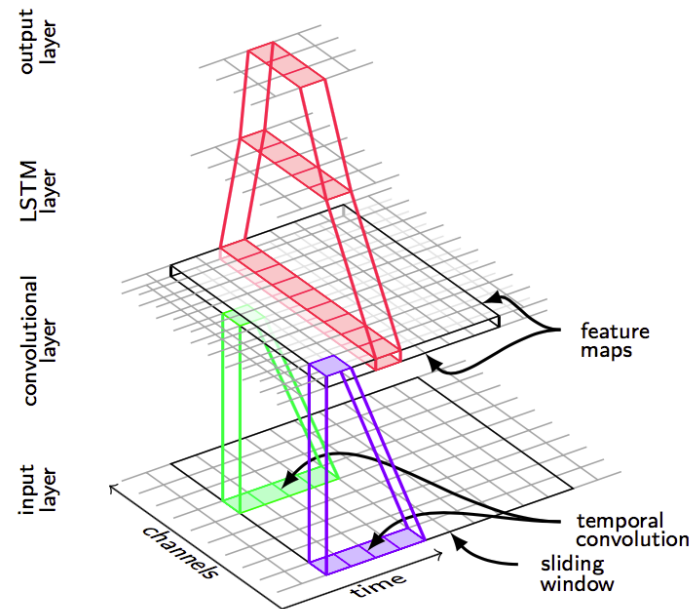
- merging raw recordings to one file
- reducing total number of classes from 5 to 4, by converting “set break” to “break”. I don't know what I was thinking when introducing two different break classes...
- converting the first and last two time steps of every squat repetition to “break”. Earlier models often counted squats, when I actually didn't do anything. This fixed it.

Choosing a model

I intended to write my master thesis in human activity recognition (HAR), but I didn't find a supervisor. Anyway I could use some of the insights from my thesis proposal (<https://github.com/Lausbert/Exermote/blob/master/ExermotePreprocessingAndTraining/MasterThesisProposal/master%20thesis%20proposal.pdf>). The following table is an excerpt from this proposal.

Table 1: Examples of fitness activity recognition in research (Sensor abbreviations: accelerometer, gyroscope, magnetometer: 'aclm', 'gyr', 'mag'. Evaluation metrics abbreviations: F-measure, accuracy, precision, recall: 'F', 'acc', 'prec', 'rec'. Posture ab-

As you can see in the last row DeepConvLSTM Neural Networks were already tested by Francisco Javier Ordóñez and Daniel Roggen for recognizing activities of daily living. Their approach (<https://github.com/sussexwearlab/DeepConvLSTM>) and their results (<http://www.mdpi.com/1424-8220/16/1/115/html>) impressed me and so I decided to take their model and give it a try for my purpose. The model takes time sequences of raw sensor data and outputs the according exercise label. A simplified model representation looks like this:



The actual model differs in terms of layer and channel (data feature) numbers. Furthermore a higher stride and a dropout layer were added for better generalization:

```
model = Sequential([
    Conv1D(nodes_per_layer, filter_length, strides=2, activation='relu', input_shape=(timesteps, data_dim),
        name='accelerations'),
    Conv1D(nodes_per_layer, filter_length, strides=1, activation='relu'),
    LSTM(nodes_per_layer, return_sequences=True),
    LSTM(nodes_per_layer, return_sequences=False),
    Dropout(dropout),
    Dense(num_classes),
    Activation('softmax', name='scores'),
])
```

Training

The whole training procedure took place in the google cloud, since I found this wonderful tutorial (<http://liufuyang.github.io/2017/04/02/just-another-tensorflow-beginner-guide-4.html>). The machine learning framework in use was Keras with TensorFlow as backend. Many thanks to Google for 300\$ of free credits. After training hundreds of models there are still plenty left:

Credits



€177.56

Credits remaining

Out of €282.41

For training observation I used TensorBoard:

The (optimum) parameters shown below were determined during training. `timesteps` defines the sliding window length, while `timesteps_in_future` specifies which time step label should be characteristic for a sliding window. More `timesteps_in_future` would mean a higher accuracy in recognition, while it would worsen live prediction experience.

```
# training parameters
epochs = 50
batch_size = 100
validation_split = 0.2

# model parameters
dropout = 0.2
timesteps = 40
timesteps_in_future = 20
nodes_per_layer = 32
filter_length = 3
```

While training models with various input combinations, it became clear that the benefit of using the mentioned Nearables is a smaller one. Therefore I gave up on using them any longer. Additional sensors might get interesting again for recognizing one-armed exercises or exercises where only your feet and/or legs are moving.

```
X = dataset[:, [
    2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, # Device
    # 14,15,16,17,                        # Right Hand
    # 18,19,20,21,                        # Left Hand
    # 22,23,24,25,                        # Right Foot
    # 26,27,28,29,                        # Left Foot
    # 30,31,32,33,                        # Chest
    # 34,35,36,37                        # Belly
]].astype(float)
```

The highest recognition accuracy achieved on test data with only 12 data features was 95.56 %.

Since mainly first or last timesteps of a repetition were confused for a break or the other way around, this accuracy is sufficient for recognizing and counting the mentioned exercises. The best model of a training procedure was saved to google cloud bucket. The model was also exported to .pb and .mlmodel format for later use on google cloud and on iPhone.

Inference

The model is already built, so let's put it to work!

Google Cloud ML

Before WWDC 2017 and CoreML I couldn't find a proper way for doing inference directly on my iPhone. That's why I implemented my model on Google Cloud ML (<https://github.com/Lausbert/Exermote/tree/master/ExermoteInference/ExermoteMachineLearningEngine>). Acceleration Data was sent 10 times per second to the cloud, while receiving inference results in the same frequency. This worked surprisingly well! At least for one minute, then it appeared that the iPhone was blocking any further network requests. What a lucky coincidence that Apple introduced CoreML a short time later 😊

CoreML

Since I already exported the model as .mlmodel file, implementing it was quite easy. The interesting line below is `let predictionOutput = try _predictionModel.prediction(input: input)`, because that is where calculation is done. Actually initialization of model inputs was the hardest part and as you can see below it is done in a not very swift way. Let's hope that this is due the beta status of CoreML.


```

func makePredictionRequest(evaluationStep: EvaluationStep) {
    let data = _currentScaledMotionArrays.reduce([], +)
    do {
        let accelerationsMultiArray = try MLMultiArray(shape:[40,1,12],
            dataType:MLMultiArrayDataType.double)
        for (index, element) in data.enumerated() {
            accelerationsMultiArray[index] = NSNumber(value: element)
        }
        let hiddenStatesMultiArray = try MLMultiArray(shape: [32],
            dataType: MLMultiArrayDataType.double)
        for index in 0..32 {
            hiddenStatesMultiArray[index] = NSNumber(integerLiteral: 0)
        }
        let input = PredictionModelInput(accelerations: accelerationsMultiArray,
y,
                                lstm_1_h_in: hiddenStatesMultiArray,
                                lstm_1_c_in: hiddenStatesMultiArray,
                                lstm_2_h_in: hiddenStatesMultiArray,
                                lstm_2_c_in: hiddenStatesMultiArray)
        let predictionOutput = try _predictionModel.prediction(input: input)
        if let scores = [predictionOutput.scores[0],
                        predictionOutput.scores[1],
                        predictionOutput.scores[2],
                        predictionOutput.scores[3]] as? [Double] {
            evaluationStep.exercise = decodePredictionRequest(scores: scores)
        } else {
            print("Could not cast predictionOutput.scores to [Double].")
        }
    }
    catch {
        print(error.localizedDescription)
        self.stopPrediction()
    }
}

```

The result of my project is a pretty stable exercise recognizer! 😊

[convolutional](http://lausbert.com/tag/convolutional/) (<http://lausbert.com/tag/convolutional/>) [coreml](http://lausbert.com/tag/coreml/) (<http://lausbert.com/tag/coreml/>)

[exermote](http://lausbert.com/tag/exermote/) (<http://lausbert.com/tag/exermote/>) [firebase](http://lausbert.com/tag/firebase/) (<http://lausbert.com/tag/firebase/>)

[fitness](http://lausbert.com/tag/fitness/) (<http://lausbert.com/tag/fitness/>) [googlecloudml](http://lausbert.com/tag/googlecloudml/) (<http://lausbert.com/tag/googlecloudml/>)

[ios11](http://lausbert.com/tag/ios11/) (<http://lausbert.com/tag/ios11/>) [keras](http://lausbert.com/tag/keras/) (<http://lausbert.com/tag/keras/>)

[LSTM \(http://lausbert.com/tag/lstm/\)](http://lausbert.com/tag/lstm/) [Neural Network \(http://lausbert.com/tag/neural-network/\)](http://lausbert.com/tag/neural-network/)

[tensorflow \(http://lausbert.com/tag/tensorflow/\)](http://lausbert.com/tag/tensorflow/)

2 thoughts to “Exermote: Building a fitness tracker with Convolutional LSTM Neural Networks”



ANKIT

11. September 2017 at 12:56 (<http://lausbert.com/2017/08/03/exermote/#comment-3>)

So cool!! Great job!

REPLY ([HTTP://LAUSBERT.COM/2017/08/03/EXERMOTE/?REPLYTOCOM=3#RESPOND](http://lausbert.com/2017/08/03/exermote/?replytocom=3#respond))



LAUSBERT

11. September 2017 at 13:12 (<http://lausbert.com/2017/08/03/exermote/#comment-4>)

Thank you! Good to hear 😊

REPLY ([HTTP://LAUSBERT.COM/2017/08/03/EXERMOTE/?REPLYTOCOM=4#RESPOND](http://lausbert.com/2017/08/03/exermote/?replytocom=4#respond))

LEAVE A REPLY

Your email address will not be published.

Comment

=====

Name

Email

Website

POST COMMENT

FOLLOW ME