Snorre Børtnes

Segmentation of repetitions and sets in weight lifting workouts with LSTM networks

Master's thesis, spring 2019

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

Segmenting time series data is an essential problem in human activity recognition. An increasing area of interest in this field has been automatically recognizing and logging workouts by using data from wearable sensors like smartwatches and phones. Using deep learning technology in these devices requires models that are computationally efficient to conserve power. This thesis will focus on segmenting sets and repetitions from a workout of weight lifting exercises. These segments can further be used to count sets and repetitions from performed exercises during a workout. Extracting the number and type of these events are significantly more straightforward to do on a segmented sequence, compared to algorithms applied directly on the raw data. To solve this task, a LSTM based model is proposed that segments both the exercise activity and the repetitions at the same time. Data from full workouts, performed in a real-life scenario, is collected to perform experiments and show the performance of this model. The model can segment the time series in such a way that minimal filtering would be required in order to apply a simple algorithm to count repetitions.

Sammendrag

Segmentering av tidsdata er et sentralt problem i menneskelig aktivitetsgjenkjenning. Et område med økende interesse er å automatisk kunne generere sammendrag av treningsøkter ved hjelp av data fra smarttelefoner og smartklokker. Å bruke dyp læring i disse elektroniske enhetene krever at modellene er er effektive slik at enhetene skal ha et minimal strømforbruk. Denne oppgaven fokuserer på å segmentere sett og repetisjoner fra styrkeøvelser utført i en treningsøkt. De segmenterte delene kan videre brukes til å telle antall sett og repetisjoner for øvelsene utført i økten. Å finne antall sett og repetisjoner er vesentlig enklere å gjøre på en sekvens av disse segmentene enn det er å designe algortimer som brukes direkte på rådataen. For å løse segmenteringsoppgaven blir det utviklet en LSTM basert modell som samtidig segmenterer ut øvelsene og repetisjonene som utføres. Data blir innsamlet fra treningsøkter som utføres så realistisk som mulig. Modellen blir trent og testet på denne dataen for å teste ytelsen. Modellen klarer å segmentere tidsseriene på en måte som gjør at minimal filtrering ville være nødvendig for å kunne bruke en simpel algoritme for å telle repetisjonene.

Preface

This project was performed as a Master's thesis at the Department of Computer Science at NTNU. I would like to thank my supervisor from NTNU, Keith L. Downing.

Snorre Børtnes
Trondheim, June 7, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter motivates the purpose and goal of this thesis. Section 1.1 gives a brief explanation of the driving forces that motivated the research in this thesis. In Section 1.2 it establishes a research goal and research questions to help guide the research towards efficiently and accurately segment exercise activities and repetitions in time series data. Then, Section 1.4 provides an overview of the main chapters in this thesis and their contents.

1.1 Background and Motivation

Fitness tracking devices and applications are becoming a more central part of many peoples workout regiment. Logging these workouts manually is tedious and is required to do consistency over longer periods of time in order to provide valuable information to the user. Smartphones or wearable devices such as smartwatches contain sophisticated and a wide selection of sensors capable of recording high frequency time series data. These devices makes it possible to detect exercise activities such as running, cycling and swimming. They can automatically create entries in a workout dairy, only requiring the user to wear the device during the activity.

Fitness trackers logging weight lifting workouts are however less common. Endurance based activities are easy to recognize in a datastream as they consist of a movement pattern that is continuously performed over longer periods of time. Logging these workouts does in many cases only consist of finding the start and the end of the activity. The activities performed during a weight lifting workout are significantly different, and so is the requirement of details in the logging of these activities. The lifting activities in such workouts are short events compared to total time spent on the workout. Logging such a workout consists of counting

the number of sets and repetitions performed, for each exercise performed.

Deep learning has proved to be a great tool in many fields due to its ability to automatically extract features from raw sensor data. It does, however, often require significantly more computational resources than traditional methods. To use this technology in devices like smartphones and smartwatches, power efficient and computationally efficient models have to be used. This thesis will attempt to create a recurrent deep learning model able to segment exercise activities and repetitions from time series of weight lifting workouts.

1.2 Research Goal and Questions

To reach the goal of efficiently and accurately segmenting time series from workouts, a research goal and a set of research questions were made to guide the research. It is essential to get an overview of what work has already been done to solve these tasks, and that can inspire and guide the research of this thesis.

Research Goal *Segment motion data from weightlifting exercises such that sets and repetitions can be recognized.*

Automating recognizing sets and repetitions in time series data from weight lifting workouts consist of segmenting out sets and repetitions from time series data. The goal is to find a way to segment time series data from weight lifting workouts into a set of predefined categories.

To create more fundamental tasks that can be solved to accomplish the goal, two research questions are formulated. These questions are the following:

Research question 1 *What are state of the art methods for segmenting time series data of workout activities?*

Research question 2 *How can repetitions of a given exercise be recognized?*

1.3 Research Method

To address the research questions stated in 1.2, a structured literature review will be carried out to find state-of-the-art methods for worn sensor-based exercise and repetition recognition. A model will be designed and tested with experiments to quantify its performance. Data for these experiments will be collected from various participants performing weight lifting workouts. The reason for collecting a dataset specifically for this thesis is to have full control of the data pipeline such that the collected data is as realistic as possible. The data collected will be from a sensor worn by the participants while performing the workout. The literature

review and collected data will then jointly be the basis of which the the model and the experiment is designed. The results from the experiments will then attempt to answer the research questions.

1.4 Thesis Structure

This section describes the remaining structure of the thesis, and provides a brief summary of the content in the chapters.

- **Chapter 2: Background Theory And Motivation** provides an introduction to the theory required to understand the model used for the experiments as well as the meaning of the results. It also explains the process of conducting a structured literature review and how the review was performed for this thesis. It presents the related work found from this review and how the research questions stated in 1.2 can be addressed as well as motivated the work of this thesis.
- **Chapter 4: Methodology** presents the approach taken to collect data, and how to was preprocessed. It justifies the choice of a recurrent model to solve the task of segmenting activity data both by activity type and repetitions. Then, the method used for training and evaluating the proposed model is presented.
- **Chapter 5: Experiments and Results** presents the experiment that was conducted, and describes the hyperparameter values that were used for the model in the experiment. It then presents the results of the experiment, and an analysis is performed to evaluate these results.
- **Chapter 6: Conclusion** discusses the validity of the results, potential sources of errors or bias as well as shares additional thoughts on how to improve the data collection process. For future work, design choices that might provide better results, as well as other ways to improve the performance and applicability of the method used in this project are discussed.

Chapter 2

Background Theory

This chapter present the basic theory needed to understand the methodology and experiments conducted in this thesis. Section 2.1.1 briefly presents the difference between artificial intelligence, machine learning and deep learning. Section 2.1.2 explains the basic function of neural networks while Section 2.1.3 talks about how these networks often needs to be regularized in order to generalize to new data. Then, in Section 2.1.3, some theory about recurrent neural networks are presented.

2.1 Background Theory

2.1.1 Machine Learning

While humans through the times have been preoccupied with understanding biological intelligence, the field of artificial intelligence (AI) concerns building intelligent entities. AI is a new field relative to many other scientific fields, emerging shortly after the second world war. Today it is a growing field with active research and practical applications in many domains. In Russell and Norvig [2016] they explore AI by looking at the rational agent, which they define as an agent that acts to achieve the best expected outcome given its perceptions.

A sub-field of AI with much traction these days is Machine Learning. Machine learning concerns learning by generalizing from examples. This is done by learning a function from a collection of input-output pairs that try to predict the outputs of previously unseen pairs correctly. For a classification problem, the output is a subset of predetermined categories, while a regression problem has continuous output values. Enabling a rational agent to learn and improve allows it to react to changes over time that cannot be anticipated when they

are designed. It also allows for solving some problems that humans solve intuitively, but human programmers might have a hard time decomposing and solving programmatically.

Simple machine learning algorithms depend strongly on the representation of the data provided to them[Goodfellow et al., 2016]. Representations consist of several features describing the data. While many tasks can be solved successfully by designing the right set of features and giving them as input to a simple machine learning algorithm, deciding what features to use can be a cumbersome process. It can be challenging to know what features should be extracted and how they should be composed. Representation learning considers approaches where machine learning is used to discover both the mapping from representation to output and the representation itself[Goodfellow et al., 2016].

The field of deep learning is again a sub-field of representation learning. The solution proposed by deep learning is to allow computers to learn to represent the world hierarchically. Hierarchies of abstractions are learned where more complicated abstractions are defined in terms of simpler abstractions. These abstractions can be learned from raw data, which removes the need for formal specification and feature engineering by humans.

2.1.2 Neural networks

An example of a deep learning model is a neural network. Neural networks were a result of loose inspiration by neuroscience and the hypothesis that mental activity mainly consists of electrochemical activity in brain neurons [Russell and Norvig, 2016]. A neural network defines a function $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$ where $\hat{\boldsymbol{y}}$ is the network output, $\boldsymbol{x}$ is the network input and $\boldsymbol{\theta}$ are the parameters learned by the network to best approximate the training data.

A neural network consists of interconnected layers of computational units. Each unit combines its inputs $\boldsymbol{x_u}$ with their associated weights $\boldsymbol{w_u}$, a constant bias term $\boldsymbol{b_u}$ and an activation function $f_u(\boldsymbol{z})$ into a single output or activation $\boldsymbol{y_u} = f_u(\boldsymbol{x_u} \odot \boldsymbol{w_u} + b_u)$. The features from the data are the inputs to the first layer. In most network architectures, the subsequent layers get their inputs from the activations of the preceding layer. The first layer is known as the input layer, the last layer as the output layer, and the layers of units in between are known as hidden layers. The number of layers gives the depth of the network, and the term "deep learning" arises from this terminology[Goodfellow et al., 2016]. Each layer outputs an encoding of the input that is then given as input to the next layer. In feedforward neural networks (FFNN), the information flows from the input $\boldsymbol{x}$ to the input layer, through to the subsequent hidden layers and finally to the network output $\hat{\boldsymbol{y}}$ with no feedback connections. More complex versions of neural networks allow feedback connections and are called recurrent neural

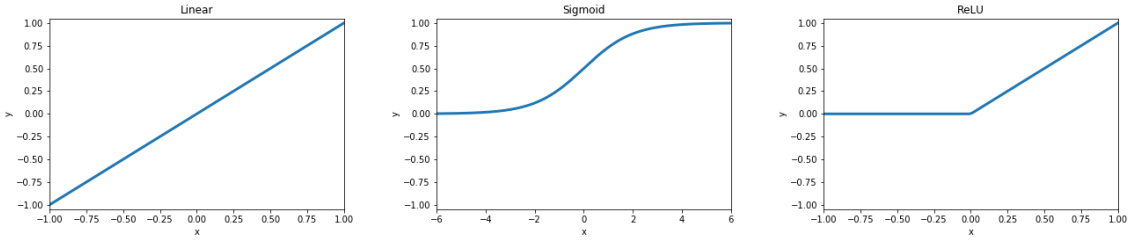


Figure 2.1: Three popular activation functions. From left: Linear activation, sigmoid activation and ReLU activation

networks (RNNs).

Several different activation functions can be used in the network units. Popular choices are sigmoids, tanh and the rectified linear unit (ReLU). Another possible activation function is the identity function $y = x$. If the identity function is used as the activation function in the entire network, all the network calculates is a linear combination of its inputs. The other mentioned activation functions add non-linearities to the network, which increase the expressive power substantially.

Some popular activation functions are presented visually in figure 2.1.

During training, the network output $\hat{\boldsymbol{y}}$ is compared to the labeled training output $\boldsymbol{y}$. Any deviation of $\hat{\boldsymbol{y}}$ from $\boldsymbol{y}$ attributes to the cost or loss of the model. The choice of the loss function is an vital aspect of the design of a deep neural network, and its specifics are connected to the choice of activation function in the last layer. Most modern neural networks are trained using maximum likelihood estimation, where the likelihood $P(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ is maximized for all samples in the dataset jointly by finding the optimal settings for $\boldsymbol{\theta}$. Maximizing the likelihood is equivalent to minimizing the mean squared error (MSE) in the case where you have a single linear output unit that produces $\hat{y}_i = \boldsymbol{w}^T\boldsymbol{h}_i + b$ for the m training examples in $\boldsymbol{X}$ with $\boldsymbol{h}_i$ as the encoding of $\boldsymbol{x}_i$ in the last hidden layer [Goodfellow et al., 2016, p. 134]. MSE is a loss function often used with regression tasks.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{2.1}$$

The learning in a neural network takes the form of updating the weights and biases in the network to minimize the loss function. The loss is propagated through the model to update all weights and biases in the direction that makes the loss decrease. For a given weight, the update that makes the loss decrease the most is the update in the opposite direction of the gradient of the loss in

regards to the weights. Since the gradient specifies the direction in which the error increases the most, the parameters are changed in the opposite direction. How much the weights are changed depends on the size of the gradient and a parameter known as the learning rate or step size. Thus the update to a given weight w given a learning rate η and loss L is given by

$$\Delta w = -\eta \frac{\partial L}{\partial w} \tag{2.2}$$

The method for computing the gradient for individual weights is known as the back-propagation algorithm[Rumelhart et al., 1986], while the basis algorithm for updating the weights is the gradient descent algorithm. The basis of the back-propagation algorithm is the chain rule from calculus. Given the example shallow feedforward neural net $\hat{y} = f(x_1, x_2; \boldsymbol{\theta}) = f_a(w_1 x_1 + w_2 x_2 + b)$, a general loss function $L(\hat{y})$ and activation function f_a, the loss propagates to the weight w_1 the following way:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1} = L'(\hat{y}) \cdot f_a'(w_1 x_1 + w_2 x_2 + b) \cdot x_1 \tag{2.3}$$

The z in the example above is the intermediate result $z = w_1 x_1 + w_2 x_2 + b$. This becomes more complicated for more larger network structures, but the principle is the same.

A problem that can come up when doing back-propagation for very deep network structures is vanishing or exploding gradients. This comes up for deep structures as the calculation of the gradients will involve the multiplication of many factors. It can thus blow up if the factors are large or vanish if the factors are small. Vanishing gradients make learning difficult as the direction to modify the parameters becomes unknown. This is especially a problem for the first layers as the information from the computed error vanishes as it propagates backwards through the network. Exploding gradients make the learning unstable as the updates to the weights will be huge. This might make the weights jump around between updates and fail to converge. These problems are particularly prominent in RNNs when the same weights $\boldsymbol{W}$ are applied at each time step of a long temporal sequence.

The non-linearities introduced to the network to enhance expressive power lead to the loss function being non-convex. This makes the training procedure sensitive to the initial network parameters as the loss can decrease to different local minima, or in some cases not converge at all. The initial point can also affect the generalization of the model. Typically, biases are initialized to small constants while the weights are set randomly[Goodfellow et al., 2016, p. 302]. Randomly choosing the initial weights breaks symmetry in the learning, and enables the individual units of a layer to learn different features. Often a Gaussian or uniform distribution is used to draw the weights.

Training in a neural network is done by feeding training examples $\boldsymbol{X}$ through the network, comparing the model output $\hat{\boldsymbol{Y}}$ to the target output $\boldsymbol{Y}$, calculating and back-propagating the loss and updating the weights. Different choices can be made of how many examples to feed through the model and accumulating loss before updating the weights. The extreme endpoints are stochastic gradient descent and batch gradient descent. While batch gradient descent computes the updates to the weights using the whole dataset, stochastic gradient descent does it for a subset of the dataset at a time. The subset of samples used for one update is called a mini-batch. A pass through all of the training data is known as an epoch, and usually, many epochs are needed.

Other design choices when constructing a deep neural network include the number of layers, the number of units per layer, and the overall network structure. In fully connected networks, every node in the preceding layer forwards its output to every node in the next layer. Other popular architectures are convolutional neural networks(CNNs) and RNNs. Gradient descent is one of many choices of optimization algorithms for neural networks, where extensions like Adam [Kingma and Ba, 2014] and AdaGrad [Duchi et al., 2011] exist that extend the simple gradient descent algorithm with features like momentum and adaptive learning rates. There is no consensus on which optimizer to prefer [Goodfellow et al., 2016, p 309].

2.1.3 Regularization techniques

The real challenge in machine learning is not to train a model that makes perfect predictions for the training data, but to train a model that generalizes to unseen samples. A range of methods address this issue and aim to increase test performance, sometimes at the expense of training performance. These methods are known as regularization techniques.

One of the simplest and most common regularization techniques is the L^2 weight penalty which is commonly known as weight decay. It is parameterised by the non-negative coefficient λ and adds the term $\frac{1}{2}\lambda||\boldsymbol{w}||_2^2$ to the loss function, which penalizes large weights.

Another option is to use L^1 regularization. It also penalizes large weights, but by the absolute value instead of the square as in L^2. Formally, $||w||_1 = \sum_i(|\boldsymbol{w}_i|)$ is added to the loss function. L^1 loss leads to sparse weights as many weights will be forced towards zero.

Dropout is another regularization technique. It involves randomly removing non-output nodes during training. During inference, all nodes are kept. The weights going out of a node are multiplied by the probability of the node being kept during training if using what is called the weight scaling inference rule. A useful characteristic of the technique is that it imposes few restrictions on the

architecture of the model. It is computationally cheap per step but reduces the capacity of the model, which leads to needing a bigger model and might require more iterations to train.

By stochastically masking nodes, the model is forced to learn more robust features that are good in many contexts, as different nodes co-occur during training iterations. The application of masking of the hidden nodes can be seen as an intelligent and adaptive filtering of the information in the input [Goodfellow et al., 2016].

RNNs and LSTMs

In contrast to predictions made from a single data sample, there also exist problems in which the temporal correlation between data samples are important. Examples of these kinds of problems are speech recognition, machine translation, processing of audio and video, and time series forecasting. All these problems contain sequential data. Sequential data is data on the form $\{x^{(t)}\}_{t=0}^{T}$, where $x^{(i)}$ is dependent on values prior to i.

Recurrent neural networks (RNNs) are a set of neural networks that specialize in capturing temporal behavior in sequential data. To capture these patterns, RNNs need something that FFNNs do not have, which is memory. This memory needs to represent the essential information from the past sequence processed by the network. It can use this memory, combined with the information at the current time step to predict the output values for the next time steps.

RNNs keep a cell state $\mathbf{C}$ of fixed length to represent memory, similar to how the output of hidden layers in FFNNs represent features at a given layer. The state changes throughout the sequence as new input is processed. The state at a given time in the sequence is denoted either $\mathbf{C}^{(t)}$ or $\mathbf{h}^{(t)}$ in literature. The network processes a sequence of values $\mathbf{x}^{(i)}$, and predicts values $\mathbf{o}^{(i)}$. The input values at $\mathbf{x}^{(t)}$ are passed through a layer with weights $\mathbf{U}$ and the previous state $\mathbf{C}^{(t-1)}$ is passed through a layer with weights $\mathbf{W}$, to extract features used to update the state of the network $\mathbf{C}^{(t)}$. This state is used to predict values $\mathbf{o}^{(t)}$ by passing the state through a layer with weights $\mathbf{V}$. A diagram of how the inputs, layers and outputs are arranged in a recurrent model is illustrated in Figure 2.2 If the task is to do time series forecasting, $\mathbf{x}^{(t)}$ would be the value of the sequence at the current time step, and the prediction $\mathbf{o}^{(t)}$ could attempt to predict the value $\boldsymbol{x}^{(t+1)}$.

The cell state needs to hold information about the temporal features observed in the past sequence of values. The state of the cell works as a lossy and compact summary of the past. Since values of fixed length represent the state of the network, information about the past sequence must eventually give way to information from more recent values in the sequence to be stored. Hence, an

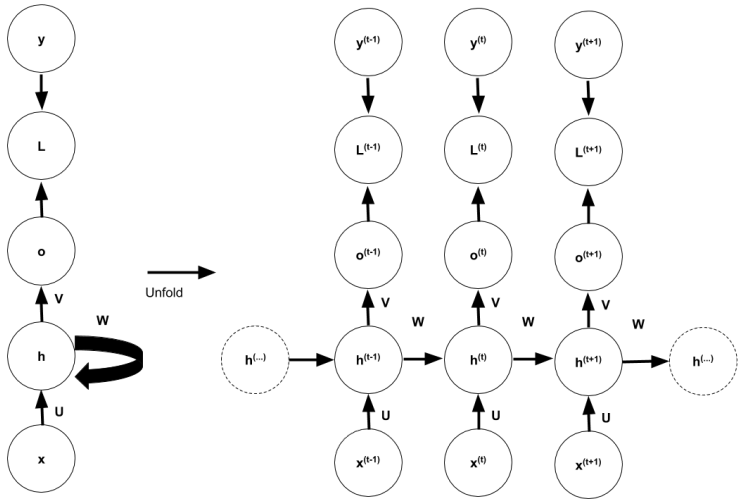


Figure 2.2: Unrolled RNN. The diagram shows the relation of inputs, weights, outputs and target values, as well as the information flow

RNN must learn to create a compact representation of observed features, in order to produce outputs that rely on patterns with long term dependencies in the sequence.

RNNs can be seen as a computational graph that contains cycles. The cycles go from the cell state of the previous time step to the next. One of the advantages of having a network with a cycle is parameter sharing. This means that it is possible to train one model with one set of weights that can process sequences of varying lengths.

For inference and backpropagation, the cycled graph is unfolded. This is done by copying the network k times for an input sequence of length k, where all the weights **U**, **W**, **V** are the same for each copy. Each network is connected to the state of the previous through the layer with weights **W**. Each copy takes its own input value from the sequence. A RNN, both as a cycled graph and unfolded, can be seen in Figure 2.2 as well as how the weights, inputs, and outputs are organized. By unfolding the network, it is possible to use the backpropagation algorithm. This backpropagation works backwards through time through weights **W**, and at each copy, a loss is calculated by comparing the output values to the ground truth. Computing gradients for the layers in each copy of the network will result in different updates to be applied in each copy even though a layer shares the same set of weights in all the copies. Since the purpose is to create one model, with one set of weights for each layer, the gradients must be aggregated,

for example by averaging, to produce one set of gradients applied to the layers.

Vanilla RNN networks have an architecture that makes capturing long term dependencies in sequences challenging. Features from the most recent values in the sequence quickly overwrite the features observed in earlier observations. Context is essential when predicting in the temporal domain, and with the memory of past values quickly fading, vanilla RNNs struggle to solve problems where long term dependencies are required to make predictions.

Solving the problem of remembering features needed for long term dependencies in a sequence was done with a gated architecture proposed by Hochreiter and Schmidhuber [1997]. This architecture was called LSTM (Long-Short Term Memory) and significantly contributed to allowing the cell state to hold and retain features extracted at different timescales. The architecture of an LSTM cell is shown in Figure 2.3. The main contributions of LSTMs are the way the gates modify the state of the cell. The gate structure allows the network to learn how much to add and how much to delete from the cell state based on features extracted from the previous state and the current input. By letting the output from the gates pass through a sigmoid activation function, a filter that can be applied over a cell state is created. It decides how much of the information in the cell state to keep since the sigmoid will output values in the range [0, 1].

The activations from the gates are applied by pointwise multiplication. The input gate filters the inputs, the previous cell state is filtered by the forget gate, and they are then combined by pointwise addition in order to create the new cell state. The new cell state is filtered by the output gate to create the output, and then this output, as well as the next input, is fed into the LSTM cell for the next time step.

The purpose of the input gate is to incorporate new information from the data being processed into the cell state of the LSTM cell, which works like the cell's memory. The forget gate has the opposite functionality, which is removing information from the cell state. Lastly, the output gate determines how the cell state is used to produce an output from the cell. All these weights consist of a fully connected layer, whose weights are adjusted during training through backpropagation. How the different gates work is therefore dependent on the training process and the data mapping that is being learned.

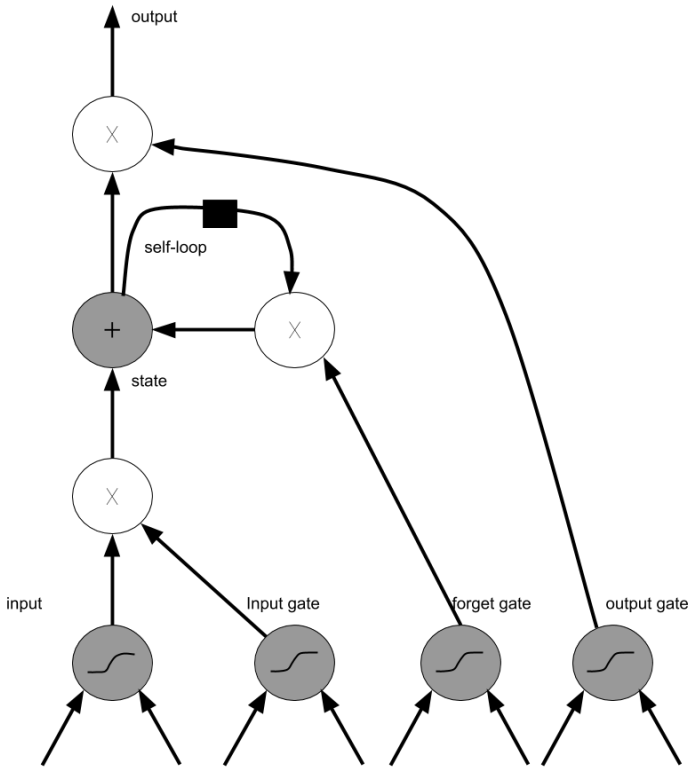


Figure 2.3: The architecture of a LSTM cell. The arrows indicate the flow of information. The two incoming arrows to the input and forget gate is the input and previous output. The output from the gray circles are passed through fully connected layers. The output from the gates use sigmoid activation function, and the output from the input and the state use tanh activation function. The circles with $\times$ and $+$ apply these mathematical operations elementwise in the input to the circle.

Chapter 3

Related Work

This chapter present previous work related to this project. In Section 3.1 the method used to perform a structured literature review is presented. Then in Section 3.2 the related work found from the review is summarized and then explained how it inspired and influenced the direction of the research in this thesis.

3.1 Structured Literature Review Protocol

To find research relevant to the research questions, and establish state-of-the-art in exercise and repetition recognition, a structured literature review (SLR) was conducted. This is a systematic search performed to identify related published work on a topic to answer a set of research questions. The advantage of performing such a structured search is to help avoid biased work, identify gaps in the research literature, and identify what can be contributed to the area Kofod-Petersen [2015]. To make this work reproducible, the review protocol is included here.

Search questions

To answer the research questions (RQs) stated in 1.2, a set of search questions (SQs) was created that breaks down the RQs into more fundamental questions that can be answered by performing SLR. Answering these SQs will collectively give the answers to the RQs. The following set of SQs was designed to answer the RQs and guide the literature review:

SQ1 What methods are used for detecting activities for exercise in time series?
SQ2 How is machine learning and deep learning used in this domain?
SQ3 What are existing methods for recognizing repetitions in exercise activities?

Search terms

To search for relevant work, a set of search terms was created. The search terms are put into groups of topics that are designed to retrieve different sets of literature. By intersecting the results of the different groups, related literature to answer the SQs are found. The search terms and groups used for the search are presented in table 3.1.

	Group 1	Group 2	Group 3	Group 4	Group 5
Term 1	har	deep learning	weight lifting	repetitive	accelerometer
Term 2	motion	neural network	exercise	repetition count	gyroscope
Term 3	activity			repetition	wearable sensor

Table 3.1: The search terms used in the SLR, divided into groups that relate the terms.

Google Scholar was used as a search engine. This decision was made as Google Scholar searches across many different publication databases in one search. It also claims to rank the results by weighing where the text was published, whom it was written by, its citation history and the full text of the documents. The search was performed without including patents and citations as they were not of interest.

By intersecting the related terms to create groups, and then taking the union of these groups, the search query was created. The search returned a total of 4210 results for this query. The results were ranked in order of relevance to the query, and the top 75 articles were included in the SLR.

Inclusion criteria

Inclusion criteria (IC) are used to select the thematically relevant articles returned from the search engine using the search string composed by the search terms. This is required as the search terms often need to be general enough in order to not exclude important literature from the search results, and this results in not all the returned results being relevant.

The inclusion criteria defined to select articles was the following:

IC1 The main field of study of the article is computer science
IC2 The main topic of the article is activity recognition
IC3 The data used in the article is from a wearable sensor recording motion
IC4 The study focuses on segmenting time series of high-frequency data
IC5 The study focuses on counting or segmenting repetitive patterns in the data
IC6 Empirical results are presented in the study

In order for an article to be considered as relevant, it needed to satisfy all the inclusion criteria. To evaluate articles on these criteria, two types of screening was performed: first title and abstract screening and then a full-text screening. If it was clear that the article did not satisfy the inclusion criteria for the first screening it was discarded. If there was doubt or uncertainty about inclusion criteria being fulfilled the second type of screening was performed.

After screening the articles selected from the search query, a total of 7 articles passed the inclusion criteria and was considered relevant for the theme of the thesis. How this research is related to and motivated the work in this thesis is discusses in the next section.

3.2 Motivation

This chapter is a summary of previous research related to the project and aims to establish state-of-the-art in the area of exercise and repetition recognition. In the previous section, the approach taken to finding related work using a structured literature review is explained. The related work resulting from this procedure is presented in this section.

In Shugang Zhang and Wei [2016] they perform activity recognition real-time on sensor data from a smartwatch. They divide the data stream with a sliding window and extract features based on accumulated difference, signal magnitude area (SMA), accelerometer data mean value, variance in accelerometer and gyroscope data and trapezoidal integration area of gyroscope data. These features are used by a SVM to perform activity recognition. In the segments where an activity is found, they use an adaptive method for finding the main axis of the movement and use a gyroscope-based counting method to count wave count on the raw data.

Qi et al. [2018] first perform a recognition phase to find whether or not an activity is a free weight or non-free weight exercise. They use data from an accelerometer as well as electrocardiogram data. They extract statistical features from the time and frequency domain as well as from the ECG and use this to perform activity recognition hierarchically. The extracted features are given to

a one class support vector machine to classify whether the data comes from an activity that is using free weights or not. They then use a hidden Markov model to classify the free weight exercises further. Recognizing repetitions inside a set is performed by smoothing the accelerometer data, standardizing the axis value and defining a vertical and horizontal threshold for counting peaks.

Maheedhar et al. [2016] perform feature extraction on windows from their data streams of accelerometer and gyroscope data of popular strength workout activities. In total, 17 features were extracted, which consisted of signal magnitude features, FFT based energy features and features based on peaks and valleys in the raw sensor data. These features are given to a shallow and small artificial neural network to classify what activity is performed in the windowed sequence. Repetition recognition and counting are first based on a separate sensor axis for each of the exercises, which is sent through a low pass filter and then applying a counting algorithm that detects peaks and valleys in the filtered data.

Das et al. [2017] takes a similar approach. They use accelerometer data and derive features based on a pairwise correlation between axes, finding dominant axes by standard deviations, finding pairwise dominant axis factors, a stillness factor and a signal time period. They compared results using decision trees, k-nearest neighbor, support vector machine, and a deep neural network. The best results were obtained using the deep neural network. To recognize repetitions, they design thresholds based on standard deviation to the average values in the signal and use this to count cycles in the signal.

Morris et al. [2014] uses inertial sensors worn on participant's right hand to segment exercise from non-exercise, classify the exercise type for the periods where exercise is detected and perform counting of repetitions. In their experiments, they found a significant difference in results for data collected in a laboratory environment with instructions regarding the form and execution of the exercises by the participants. Precision and recall levels dropped from almost perfect, down to 50 % when tested on data collected in a natural environment. Their method for exercise detection was based on that data from exercises are more repetitive than non-exercises. Therefore they utilized autocorrelation, which is cross-correlation of the signal with itself, and extracted some of the features they used for classification based on peaks in the result of the autocorrelation. In addition to correlation features, they computed statistically based features similar to other mentioned related work. They did this for windows in the sequence and trained a support vector machine to differentiate exercise activity from non-exercise activity. Exercise recognition was performed extracting similar features as when detecting the type of activity, and was then fed to a multiclass support vector machine. Repetition recognition and counting were performed based on peaks in the accelerometer data, aided by defined minimum and maximum thresholds of how long a repetition normally take.

A large amount of the articles selected from the SLR used shallow machine learning techniques, or only feature engineered algorithms. This was somewhat surprising regarding the advancement of deep learning methods applied to signal processing in many fields. Traditional and shallow machine learning techniques on time series usually involves applying a sliding window to the original signal, then extracting designed features from this time window. The features are then given to a classifier, like a support vector machine, a hidden Markov model or a shallow artificial neural network. This method is used in Shugang Zhang and Wei [2016], Qi et al. [2018], Maheedhar et al. [2016], Das et al. [2017] and Morris et al. [2014]. The main differentiating factor of how segmentation or recognition is performed in this research is how the extracted features are generated. The features generated are based on auto-correlation, PCA, statistical features, FFT coefficients and by finding the main axis for the movement and emphasize this in the detection.

These articles then analyze segments of the time series where an exercise is performed to recognize repetitions. This is primarily done through signal smoothing and counting peaks and valleys in the raw data. Counting peaks is sometimes done in complex ways by regarding information about the length of the repetitions for different exercises, finding the best axis to consider and additional rule-based procedures. The problem with counting peaks is that there is a need to know how the data signal of an exercise should look like and it is dependent on the signal following this known structure This needs to be done for every exercise, and therefore require a large amount of feature engineering. This rule-based approach might also not generalize very well to varying executions of the exercises.

Only one article returned from the SLR focused on segmenting exercises and repetitions using deep learning. This work was performed by Soro et al. [2019] and uses a convolutional neural network to process the time series data. They regard the data streams as an image by letting the values from different sensor streams be the height of the image, and the sequence of these samples to be the width. This way, they can use a fixed size CNN in a sliding window approach in order to process time series data. The best architecture of the model was found with five convolutional layers and two fully connected layers. The model was found to perform best on 7-second windows on the 100 Hz data they used. They also used multiple sensors, one worn on the forearm and one on the ankle. Each data point in their time series belonged to one of the categories of the exercises they would detect. For each window of data used as input to their CNN, they output labels for multiple data points, which are later used in majority voting and then smoothing for the final category of each data point.

To segment repetitions, they reuse the same architecture they used for exercise segmentation. They did this by labeling each half of a repetition within a set

with different categories. The first half of the repetition would belong to category 0, and the second half category 1. If a deep learning model were able to translate a raw data signal into this format, counting repetitions would be performed by a simple algorithm detecting transitions between two categories in a sequence of data points, rather than trying to deal with peaks and valleys in numerical raw sensor data. Their method consisted of first using a CNN trained to find and recognize segments of the different exercises. Then they trained additional CNNs of the same architecture, for each of the exercises, to perform the binary classification on the segment found by the exercise recognition model.

The work in this thesis is inspired by Soro et al. [2019] and the method they used to segment repetitions. They addressed that their method mainly was aimed to produce the best possible results while leaving methods that require less computational resources to future work. In order to run such algorithms on an end device, like a smartphone or smartwatch, it is crucial for the data processing to be power efficient. This research addresses this gap and will try to obtain comparable results by solving the task using less computationally intensive models and a lower data frequency. Using a CNN and a sliding window require that each data point in a sequence needs to processed multiple times. This is because the CNN cannot accurately produce labels for every data point included in the window it processes since patterns both before and after a data point is valuable to determine its category. Therefore, overlap of the sliding windows is needed. A recurrent model would allow each data point to be processed once and allows for an overall lower computational complexity. Instead of relying on sliding a CNN with a high overlap over a data sequence, it will rely on the ability of recurrent models to find temporal patterns in data to classify the data points.

Using deep learning to solve the task of segmenting exercise activities and repetitions appear to be the most general and overall the most scalable approach. The advantage of using deep learning lies in automatic feature extraction. Regardless of what sensors are used, how many are used, where they are placed and what exercises are performed, deep learning will allow finding the relevant patterns and generalize given enough data.

Chapter 4

Methodology

This chapter will explain the approach used to design the system to answer the reserach questions. In Section 4.0.1 a system using segmented exercise activities and repetitions is presented as well as how these segments can be found. Section 4.0.2 presents the process used to collect data for the experiments and some basic statistics about the collected dataset. In Section 4.0.3 the architecture and the design decisions for the model used in the experiments are presented. How the collected data was preprocessed is explained in Section 4.0.4. Then, how the model was trained is presented in 4.0.5, while how the model was evaluated is presented in 4.0.6.

4.0.1 System

The overall goal of a weight lifting recognition system is to provide the data processing tools required to automatically create a transcript of a workout from raw sensor data. This transcript would consist of how many repetitions and sets of each exercise is performed during the workout. The implemented system will work towards this goal by creating segments in the data stream from the workout. These segments translate to sets and repetitions of an exercise that is being performed. A segment of a time series is a continuous sequence of data points of the same category. The system will attempt to create the segments by categorizing each data point in the time series. In a general perspective, the task is to transform a sequence of numeric data from the movements during the workout into a sequence of categories. Therefore it would be possible to recognize when a set of an exercise is performed by finding a continuous sequence of a category, or by finding a cluster of a category, in the transformed sequence.

The activities included in this project, with their id's, are the following:

- 0 : NULL / Idle
- 1 : Squats
- 2 : Standing barbell shoulderpress
- 3 : Benchpress
- 4 : Barbell curl
- 5 : Dumbbell lateral raise
- 6 : Deadlift
- 7 : Triceps rope pushdown

The NULL category is a background category used for all other activities that is not one of the other exercises.

In addition to segment the time series based on what activity is performed, the system will attempt to segment the repetitions these segments consist of. This is done by solving the problem of repetition segmentation the same way as activity segmentation. The fact that each repetition consist of two phases is used to create categories for the repetitions of each exercise. These phases consist of two different movements that are required to perform a repetition. One phase moving the weight to a different position when starting from the initial position, and one phase moving the weight back. Each of these phases, for all of the exercises, is given a class called the repetition phase. The NULL activity only has one repetition phase as it is the background category.

The repetition phases used in this project are the following:

- 0 : NULL / Idle
- 1 : Squat phase 1
- 2 : Squat phase 2
- 3 : Standing barbell shoulderpress phase 1
- 4 : Standing barbell shoulderpress phase 2
- 5 : Benchpress phase 1
- 6 : Benchpress phase 2
- 7 : Barbell curl phase 1
- 8 : Barbell curl phase 2

- 9 : Dumbbell lateral raise phase 1
- 10 : Dumbbell lateral raise phase 2
- 11 : Deadlift phase 1
- 12 : Deadlift phase 2
- 13 : Triceps rope pushdown phase 1
- 14 : Triceps rope pushdown phase 2

Each repetition consists of a segment of the first phase, followed by a segment of the second phase. A single repetition of an exercise can be recognized by finding a segment of phase 1, followed by a segment of phase 2. The advantage of this approach is that the repetition counting algorithm can be applied on on the transformed categorical sequence and be reused for all kinds of repetitive exercises. In comparison, one would have to resort to feature engineering for each type of exercise to be able to count repetitions from the raw data.

For a user ready version of an application automatically transcribing workouts, certain features would seem useful from a user perspective. One would be for users to verify that the set they just performed was classified correctly, and the number of repetitions was correct. This imposes both that the processing would have to be executed on either the smartwatch or the smartphone of the user, and the processing has to happen close to real-time. A realistic scenario would be for a user to perform a set of an exercise, rack the weights and then verify the recorded set. From the end of execution of an exercise to the time a user would expect to verify it was recorded correctly, the period can be expected to be as short as 5 or 10 seconds. Since the processing has to happen close to real-time and have low power usage, it needs to be computationally efficient.

To handle these requirements, a recurrent network seems like the right candidate for the segmentation task. The advantage of using a recurrent model over a convolutional model is the memory property of the recurrent model. This property allows the model to process each data point in the data stream only once, as it builds a memory of what it has previously seen. Using a convolutional model would require to define a window of fixed length and slide this network over the time series. A reasonable way to do this would be to predict the categories of some of the data points in the middle of this window, as the model could observe patterns both before and after the data points it was to categorize. By sliding this window across the data stream to make predictions, each data point in processed multiple times as the windows needs to be overlapping.

To expand on the research in Soro et al. [2019], which addressed future work would include efficiently applying their method, a LSTM architecture was chosen

as the model to process the data stream. Also, both activity segmentation and repetition segmentation will be performed with a single trained model, in comparison to their work where they trained a separate model to segment repetitions in the segments their activity segmentation model recognized exercises.

4.0.2 Data Collection

The exercises were chosen due to their popularity in weight lifting workouts for both experienced and inexperienced lifters. The set of exercises was limited to these exercises to reduce the time required to collect the dataset, and for participants to be able to complete all the exercises in one single workout session. Each participant performed 3 sets of each exercise, and the repetitions for the sets ranged from 8 to 12.

The sensor data was collected using an Android application named Physics Toolbox Sensor Suite on a Samsung S9 smartphone. It would be preferred to collect data using a smartwatch, but the duration of the project made using this application for data collection more feasible than writing a custom application. The smartphone was mounted on top of the participant's left forearm with a mounting device for mobile phones. How the smartphone was attached to the forearm is shown in Figure 4.1. The phone was mounted in the same orientation for all participants.

The application allowed recording time series using multiple sensors on the phone. Three types of data streams were collected. The data streams were the g-force meter, linear acceleration and gyroscope. The g-force meter and linear acceleration are almost the same, but the g-force meter measurements are offset based on the orientation of the phone relative to gravity while the linear accelerometer is only based on the acceleration of the phone. Each of these data streams consists of measurements in three dimensions. Recorded time series could be exported as CSV files to cloud storage directly from the application, which made the data extraction process fast and straightforward.

In addition to the phone used for data collection and the mounting device, a second smartphone was used as a stopwatch to collect timestamps used for labeling segments in the time series. An assistant would use the second smartphone to register these timestamps. It was desirable for participants to perform the exercises in a natural way as possible, and the participants would decide the pace of both the repetitions and break between sets. The data was collected in a fitness center, and the assistant performed the workout together with the participants by alternating sets. The participants were encouraged to perform activities they usually performed during the breaks of sets. Such activities would included loading on and off weights, drinking, walking, checking messages on their phone and other related activities. All this data was collected and labeled as the NULL

Figure 4.1: The phone was mounted on the participants left hand to create data similar to what a smartwatch would.

category.

To simplify the software needed to label segments of the sequences, the participants were asked to choose a fixed number of repetitions to perform for all 3 of the sets they performed of a single exercise. They were also asked to choose a weight they were able to control but that was still challenging, and that they would be able to change this weight between sets if it was to light or to heavy. The data collection on the phone and the stopwatch was to the best ability started simultaneously such that timestamps from the stopwatch could directly be used to segment the sequences. The sequences collected throughout the workout were split to contain 3 sets of each exercise, including the activities in the breaks between each set and the setup of the weight lifting equipment. This was done to simplify the labeling software further such that each sequence would only contain exercises of one category.

Since the goal was to label each repetition in the exercises performed, the assistant would observe the participants and register a timestamp with the stopwatch whenever the participant started performing a set, and for both turning points for each repetition. Since a single sequence consisted of 3 sets of the

Activity	Duration [min]	Repetitions	Fraction of time
Idle	473:01	0	79.0%
Squat	19:44	378	3.3%
Shoulderpress	16:21	372	2.8%
Benchpress	16:17	390	2.7%
Curl	17:51	384	3.0%
Lateral raise	18:04	378	3.0%
Deadlift	21:38	378	3.6%
Triceps pushdown	15:33	390	2.6%
Total	**598:29**	**2670**	**100%**

Table 4.1: Duration, repetition count and distribution of collected data.

same exercise, with the agreed upon number of repetitions, it was easy to label the sequences later. After the sets for a single exercise were performed, the CSV data was exported to cloud storage and the timestamps were exported to a spreadsheet. Sometimes a timestamp was missing or was registered too late or too early during the data collection process. These errors were corrected by inspecting plots of the raw data and timestamps.

Statistics about the activities in the collected dataset used in the experiments are presented in Table 4.1. This shows that most of the data collected were of the NULL category. In summary, almost 10 hours of data were collected, where approximately 80% consisted of activities performed between sets. The total repetitions in the dataset were 2670 and are evenly spread across each of the exercises.

4.0.3 Architeture

The architecture of the model used i the experiments consisted of two LSTM cells and two separate fully connected hidden and output layers for the predictions of the exercise recognition and the repetition recognition tasks. The output layers of activity and repetition recognition used one-hot encoded categories and therefore used softmax activation function on the output of nodes in this layer. In Figure 4.2, a diagram of the architecture described is shown, with the size of the vectors as the input and output for each layer.

In total, the model had 37,079 parameters where 31,360 of the parameters were from the LSTM cells and the remaining from the fully connected layers. These calculations are performed on every data point in the time series only once during predictions, which is computationally efficient compared to a convolutional neural network being applied to highly overlapping windowed subsequences in the time series. An empirical experiment of the difference in the computational load

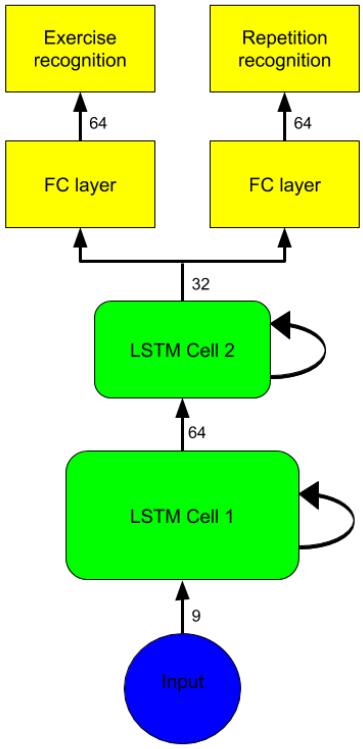


Figure 4.2: The architecture of the recurrent model. It consists of 2 LSTM cells, and for both exercise recognition and repetition recognition, there is a separate hidden layer and an output layer. The numbers next to the arrows indicate the size of the inputs and outputs to each layer.

between the models created in Soro et al. [2019] is not performed in this project. For reference, the Keras models available from the work of Soro et al. [2019] were 26 MB for the activity recognition model, and either 2 MB or 10-11 MB for the repetition recognition models, in which they trained one for each exercise. The integrated activity and repetition recognition LSTM based model used in these experiments was 175 KB.

To regularize the network, L2 loss was applied on the weights of the LSTM cells, and a dropout was applied on inputs to all layers, except the input layer and the layers for the recurrent connections of the LSTM cells.

Predicting categories for each data point in real-time is a strict limitation to impose on the model. Important information about what category a data point belongs to is also based on the data that follows it. Predicting categories real-time caused the model to be uncertain whether the categories should be the NULL category or an exercise near the end of repetitions during a set. By predicting categories real-time, the model will only have processed data up to the point in which it would make a prediction. Exercise activity is based on multiple repetitions that are performed consecutively. These repetitions abruptly stop at the end of the set, and the model would probably have some difficulty at the end of each repetition whether or not to believe the exercise is to be continued or not. This is also true for both the start of the set, the end of the set and the transition between repetition phases. If the model is to predict the transitions between categories in real-time, it must learn to anticipate what category comes next, rather than just classifying the observed sequence. The problem then starts to contain characteristics of a forecasting problem rather than a classification task.

Since the application envisioned and discussed earlier does not require results to be immediately available, it is possible to work around this problem by taking advantage of the memory in the LSTM cell. This can be done by delaying the predictions relative to the processed sequence. The introduced delay consisted of letting the model predict categories for datapoints it had seen in the recent past instead of predicting the categories real-time. The model is then allowed to take into consideration the pattern in the data both before and after the data point it is classifying. Note that this solution does not increase the computational complexity of the model as it still only processes each data point in the sequence once. This approach relies on the ability of the LSTM to remember in order to deal with the delay in predictions. A solution with 1 second of delayed predictions was chosen.

4.0.4 Data Preprocessing

The time series returned from the exported CSV did in some cases contain multiple data points with the same timestamp, and the measured values for the different data points deviated. There were up to six duplicates for data points on a single timestamp, but most of the time the duplicates was either two or three. The median value of data points with the same timestamp was calculated to remove duplicate values.

The sample rate for the collected time series in the dataset was not consistent. Although it was possible to set a custom sample rate in the application used for data collection, and this was set to 150 Hz, the output had a variable sample rate. In Figure 4.3 the time between each sample to the next within a recorded

sequence is shown, as well as a histogram showing the distribution of the sample rate. This is the data after aggregating the duplicate measurements. The top figure shows that a variable sample rate was used, even within a single recorded sequence. The histogram shows that that sample rate in most cases was either very high or in a cluster around 50 Hz (0.02 seconds between samples). The mean sample rate for all the data was approximately 120 Hz.

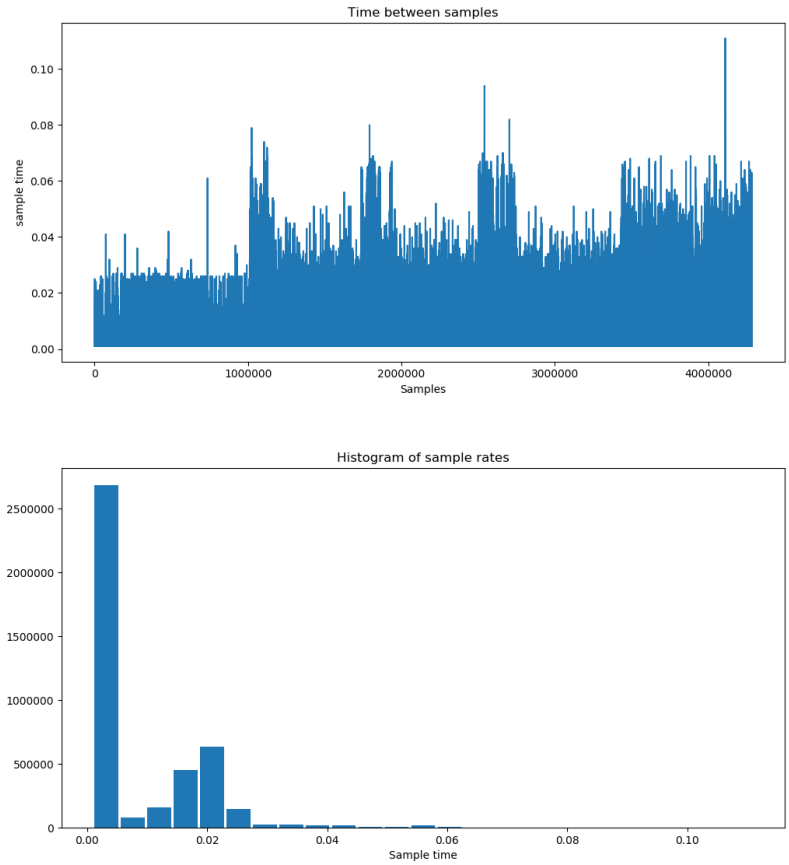


Figure 4.3: Top: The time between each sample for every sequence in the dataset. Bottom: A histogram of the sample rates in the dataset.

Some architectures, like phased LSTMs, explicitly take the time lag between inputs into account. The models tested in this project do not do this. Therefore

it is crucial to resample the time series such that the frequencies between each sample are consistent. If this is not performed, the model could interpret segments with a fast sample rate to be slowly executed movements and the opposite for segments with a low sample rate. The histogram in Figure 4.3 proves that it should be safe to resample the sequences to 33 Hz and below (approximately 0.03 time between samples) without losing information by having to interpolate missing values. A sample rate of 20 Hz was chosen for the experiments, which was believed to be sufficient for capturing data for the repetitions of the exercises since they usually span between 2-4 seconds. Since the purpose of this project was to process motion data efficiently, a lower sample rate is desirable as this leads to fewer computations in order to segment a sequence.

When resampling the sequences, they were partitioned into windows of 0.05 seconds (20 Hz), and the mean of the values within these windows was used as the final value for that timestamp. Although no smoothing was applied to the sequences later, this does have a smoothing effect. The windows that had no values were filled in using linear interpolation between neighboring values.

4.0.5 Training Phase

During training, the model was given subsequences of the collected data sequences to train on with a window length of 9 seconds. That is sequences consisting of 180 data points since 20 Hz was the frequency used when resampling. The first 2 seconds in these subsequences were used to build the state of the LSTM and did not produce predictions during training used for gradient updates. This was performed as the LSTM needs to build its state before being able to produce accurate predictions, and the networks started with a zero-state when beginning the processing of each sample. A visualization of how a state is built from an initial part of the sequence is shown in Figure 4.4. Also, the model predicted the categories for the data points it processed with a delay of 1 second. The design choice to predict the categories with delay was in Section 4.0.3. Therefore, each training sample consisted of 6 seconds of predictions used for gradient updates, and 3 seconds of data were processed before making the first prediction. Three seconds would allow the model to process the data from a little less then one repetition of an exercise to build its state if the sequence started during the execution of an exercise. The samples were created with considerable overlap, using a stride of 0.2 seconds when extracting windowed sequences for the samples from the original sequences.

Using subsequences drawn uniformly from all the sequences would result in almost 80% of the samples in each mini batch to be of the NULL category, as can be seen from the dataset distribution in Table 4.1. In preliminary experiments, the model was slow to learn the exercises performed, and would often mispredict

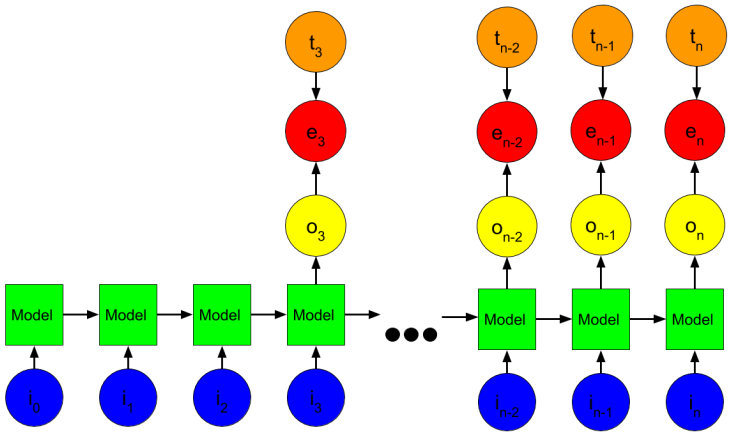


Figure 4.4: Training of a recurrent model from an empty state. The first inputs are given to the model while not producing predictions. After this set of inputs, the model produces outputs o, which are compared to target values t to calculate the error e, which is used to compute gradients.

exercise activities as the NULL category. It was therefore decided to balance the training batches with regards to the categories. This was accomplished by sorting the samples based on what categories they contained. If a sample contained one or more data points not belonging to the NULL category, it was placed together with other such samples of the same category. When creating a set of mini-batches to train on, the samples were drawn uniformly across activities from these sorted subsequences, and across the participants in the training set. This made the model learn the categories faster and achieved higher accuracy as it was not incentivized to overly emphasize the NULL category over the exercises.

4.0.6 Evaluation Phase

Evaluating the accuracy of the model was not performed with windowed subsequences like was given as training samples. Instead, the network was given the full-length sequences, that consisted of the three sets performed of each exercise including breaks and predicted the categories for every data point along the sequence. The category with the highest softmax output was used as the final prediction for each data point and this was compared with the categories obtained from data collection.

The performance of the models on the validation set was determined based on the mean accuracy of the categories of the repetition recognition, rather than the

actual accuracy for all predicted categories in the full length sequences because it was more desirable to have better recall of the sets and repetitions instead of having high precision predicting the NULL category. Since the goal is to be able to segment repetitions in such a way that it would be possible to count them the accuracy on the repetitions recognition seemed like the most valuable metric. Also, since the activity recognition should be based on repetitions, the activity recognition should contain the same segments as a set of repetition does.

It was observed that increased accuracy on the repetition recognition did not always correspond to higher accuracy for the activity recognition. A model that had lower accuracy on repetition recognition than the best model sometimes had higher accuracy in activity recognition but only by a slight amount.

Chapter 5

Experiments and Results

This chapter presents the experiments and results from the work of this thesis. In Section 5.1 the experiment conducted is explained, as well as why this type of experiment was chosen. Section 5.2 states the value of the hyperparameters used for the experiments. The experimental results are presented in Section 5.3, and the results are evaluated and analyzed in Section 5.4.

5.1 Experimental Plan

After collecting the data, the goal was to use the model proposed and shown in Figure 4.2 to learn to segment sequences from the workouts. A leave-one-subject-out (LOSO) experiment was conducted to investigate the performance of this type of network on the relatively small dataset. LOSO consists of leaving all but one of the participant in the training set and putting this participant in the validation set. The purpose is to use the training set in order to achieve as high as possible accuracy on the participant in the validation set. The dataset with the one participant left out will be referenced as the validation set to emphasize that a train/validation/test split was not performed. Although the dataset consisted of many repetitions, the repetitions performed by the same participant are very similar for a given exercise. In this sense, it is only 13 different patterns of each exercise in this dataset, one for each participant. Since the dataset consists of both large and small male and female participants at different experience levels with weightlifting, a regular training/validation/test split of the dataset would be highly sensitive to what participants are placed in the different splits, as the variance among participants were high. Doing a LOSO experiment seemed most valuable and would give insight into how well the model can learn and generalize to unseen executions of an exercise. Considering the small size of the dataset

would a k-fold split would be very sensitive to participant placement. A LOSO experiment could also indicate which participants might be outliers compared to the rest of the dataset or if specific exercises are harder to recognize than others.

5.2 Experimental Setup

The architecture used in the experiment was the one shown in Figure 4.2. The first LSTM cell had a size of 64, the second a size of 32. The fully connected layers between the output of the second LSTM cell and the output layers had a size of 64 for both the exercise recognition and the repetition recognition predictions. These fully connected layers were reused for every timestep during training and predictions, just like the layers inside the LSTM cells.

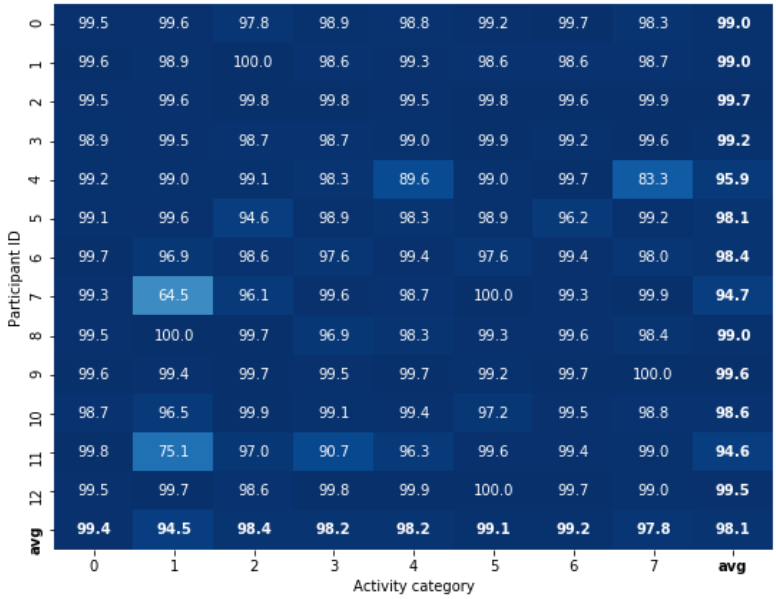
L2 loss of the recurrent and kernel weights in the LSTM cells was added with a beta value of 1×10^{-6}, while bias was not regularized. Dropout was applied to the input in all layers with a value of 20%, except the input layer for the first LSTM cell and the recurrent connections for both LSTM cells.

Each configuration of participants in the LOSO experiment was run ten times. At the start of these runs, the model was initialized from scratch with random weights. Minibatches of size 32 was used to train the models, and the samples in these batches consisted of 9-second subsequences drawn uniformly across all activities. 100 batch updates with these mini batches were performed between each time the model was evaluated on the validation set. The number of times this was performed depended on the validation accuracy increasing during training. A minimum of 50 such evaluations was performed for each run. Training of an initialized model continued until 50 evaluations on the validation set had been performed with no increase in accuracy.

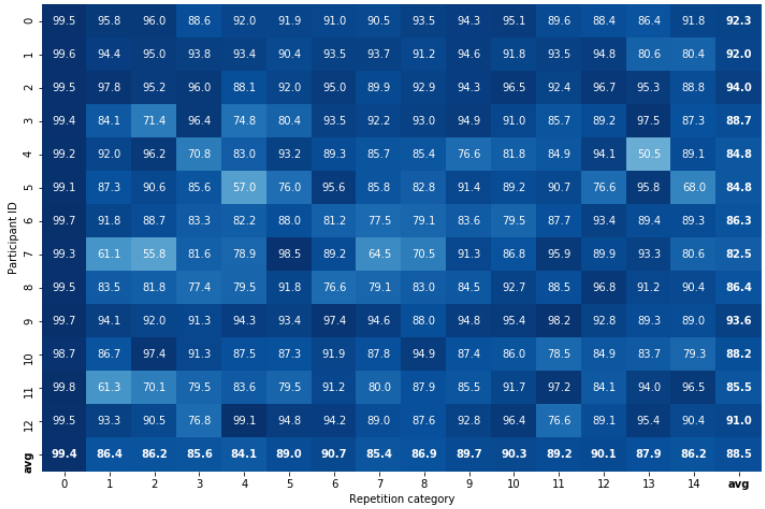
5.3 Experimental Results

In Figure 5.1, the accuracy obtained for each activity category and repetition category is presented for each participant when left in the validation set. The figure only shows the accuracy for the category that should be predicted and does not show what the false predictions were. The distribution of false predictions is shown in Figure 5.2 which is the average of the confusion matrices for all of the configurations of participants in this experiment when they were left in the validation set. The confusion matrices for all the configurations of participants in the validation set is in the appendix.

The worst results were obtained with the configuration of participant 4, 7 and 11 in the validation set. The confusion matrices for the exercise recognition and

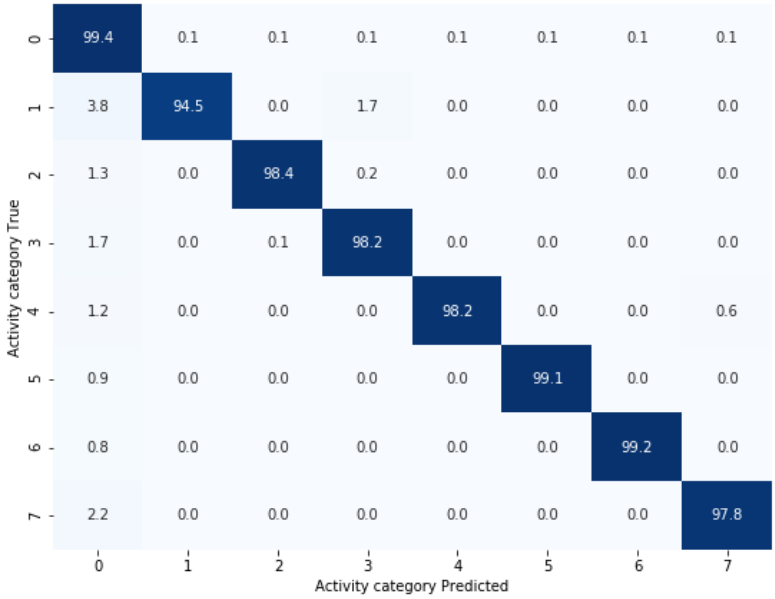


(a) Accuracies for the activity recognition.

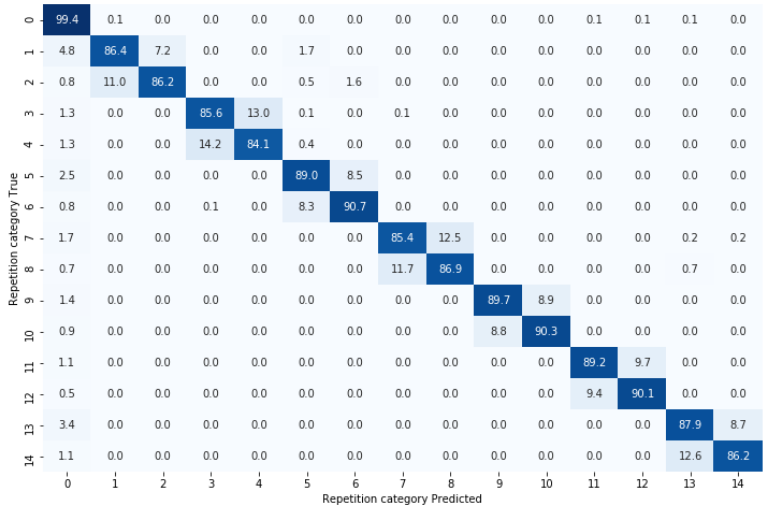


(b) Accuracies for the repetition recognition.

Figure 5.1: Accuracy for each category and participant in the LOSO experiment when that participant was in the validation set. Each entry is the percentage of how accurate the model predicted data points with that given category. The "avg" row is the average of all the participants for each exercise, and the "avg" column the average of exercises for that participant.



(a) Confusion matrix for activity recognition.



(b) Confusion matrix for repetition recognition.

Figure 5.2: Average of confusion matrices of the best models obtained for all configurations of participants in the validation set for the LOSO experiment.

the repetition recognition for these participants are presented in Figure 5.3, 5.4 and 5.5.

Some of the sets of the exercises with low accuracy are visualized in Figure 5.6, 5.6, 5.8, 5.9, 5.10, 5.11 and 5.12. The segments show data from the g-force sensor as well as predicted and true segments. The predicted segments are based on the max value of the softmax output for each data point, visualized in the bottom of the figures.

In Figure 5.6 the visualization show how the curl activity is confused with the triceps rope pushdown activity for participant 4. Figure 5.6 shows the model believing the participant stops performing the activity, and predict the NULL category for some segments in the set instead.

For participant 7 the segments are either missing the start of the setlike in Figure 5.8, confused with benchpress as shown in 5.9, or both like in 5.10.

Figure 5.11 and 5.12 show one of the sets of squat and benchpress for participant 11. In these visualizations, the model struggles to predict the start of the first repetition phase, especially for the squat activity, and predict the NULL category instead.

Since squat was the worst performing exercise, visualizations for one of the participants where the model was accurate is shown in figure 5.13. This visualization is for one of the sets of squats performed by participant 2.

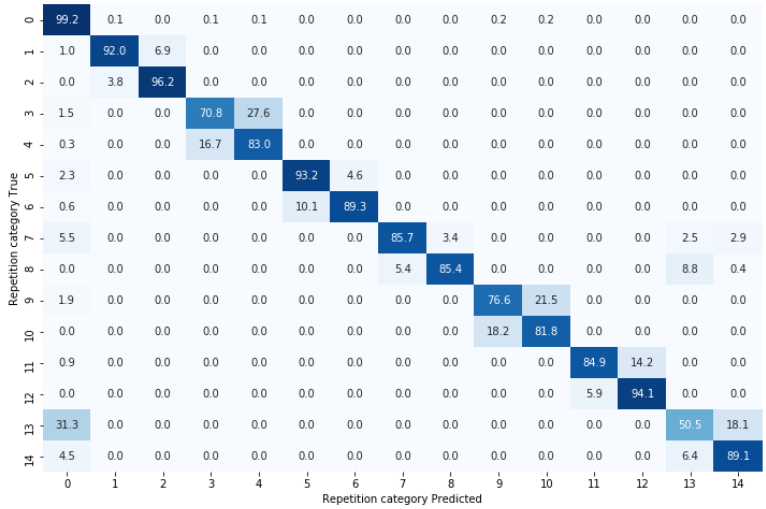
5.4 Evaluation

As can be seen from the overall accuracies for the LOSO experiment in Figure 5.1, the activity recognition was for most participant able to recognize what exercise is performed most of the time. For some participants, it was however not possible to generalize to certain activities they performed with this model.

Overall the squat activity (category 1) had the lowest accuracy for the activity segmentation, which mainly was because the model failed to obtain good results for this activity for two of the participants. The activity recognition accuracy for these two participants was only 64.5% and 75.1%, pulling the average for this activity down to 94.5%.

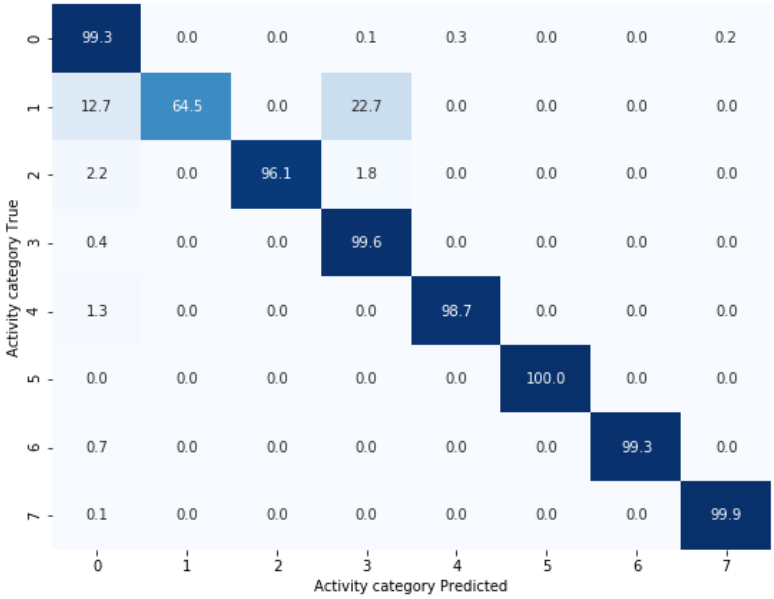
The primary source of error in predicting what activity is performed comes from miscategorizing datapoints as the NULL category. This sometimes happens because the exercise is not detected at all, like for the triceps rope pushdown exercise (activity 7) for participant 4. When the exercise was recognized, the error mainly comes from the exact start and end of an exercise segment not being predicted accurately. This fact can be seen from the average of the confusion matrices for all the participants when left in the validation set in Figure 5.1. The claim that this happened at the start and end of each set was confirmed by

(a) Confusion matrix for the activity recognition.

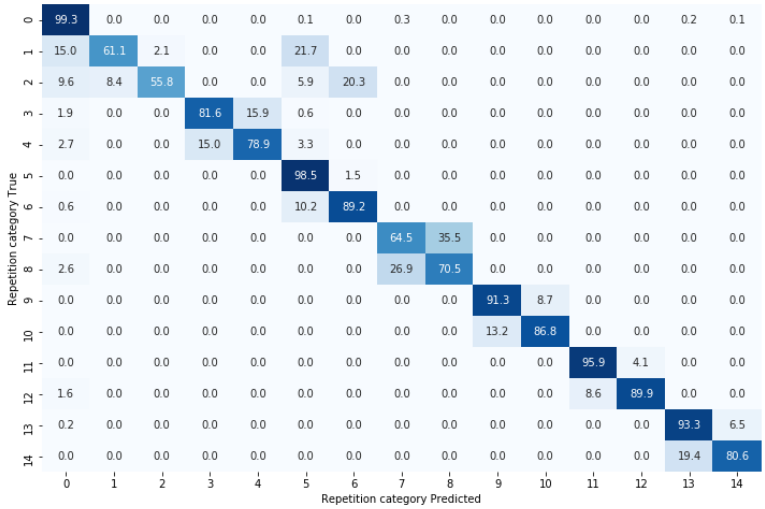


(b) Confusion matrix for the repetition recognition.

Figure 5.3: Confusion matrices for participant with id 4 when left in the validation set.

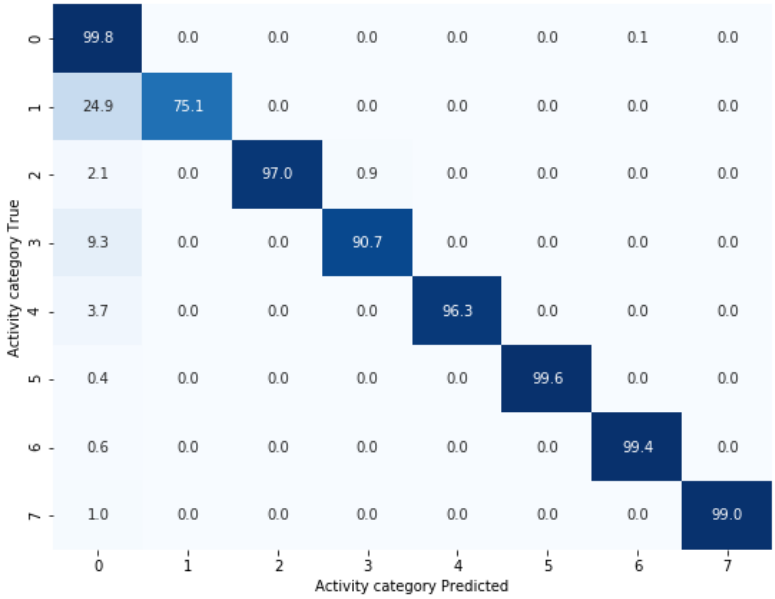


(a) Confusion matrix for the activity recognition.

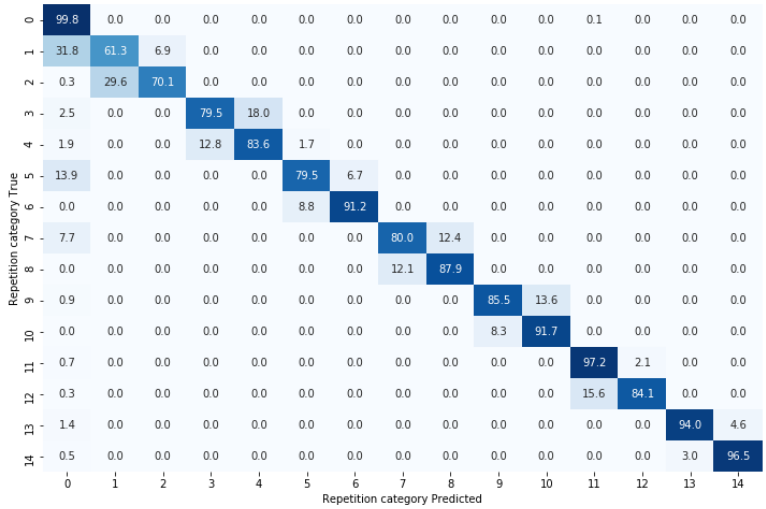


(b) Confusion matrix for the repetition recognition.

Figure 5.4: Confusion matrices for the participant with id 7 when left in the validation set.

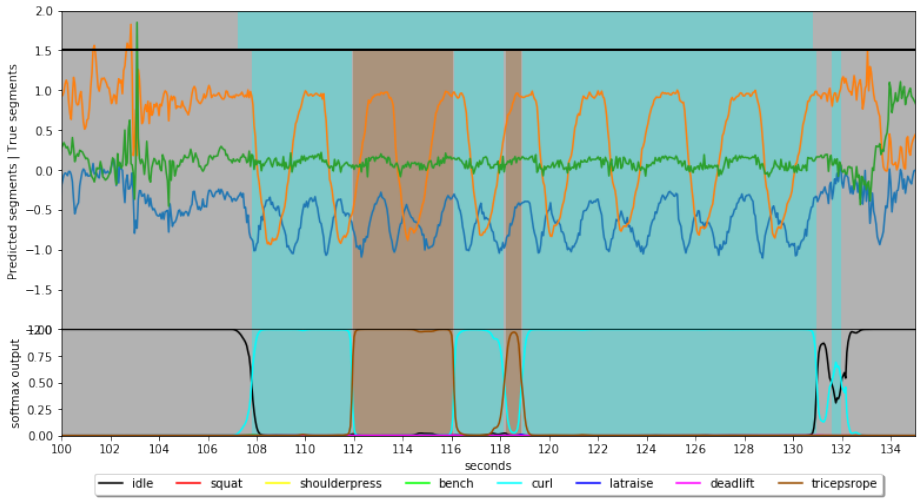


(a) Confusion matrix for the activity recognition.

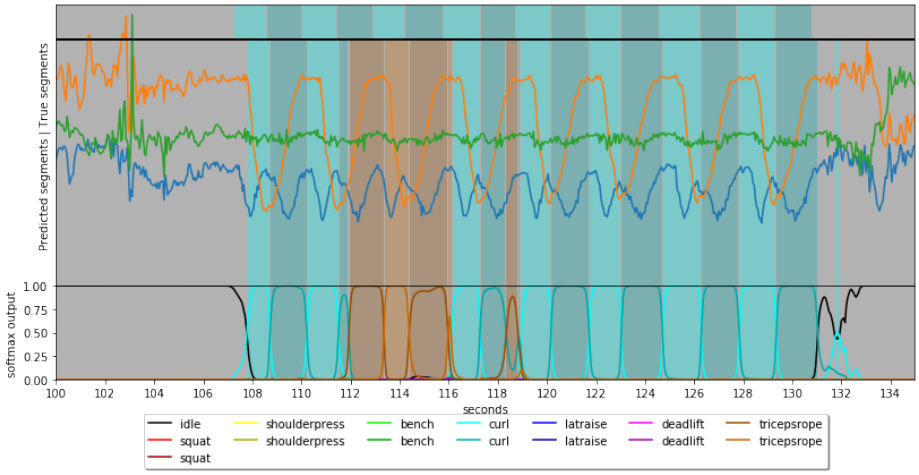


(b) Confusion matrix for the repetition recognition.

Figure 5.5: Confusion matrices for the participant with id 11 when left in validation set for the leave-one-subject-out experiment.

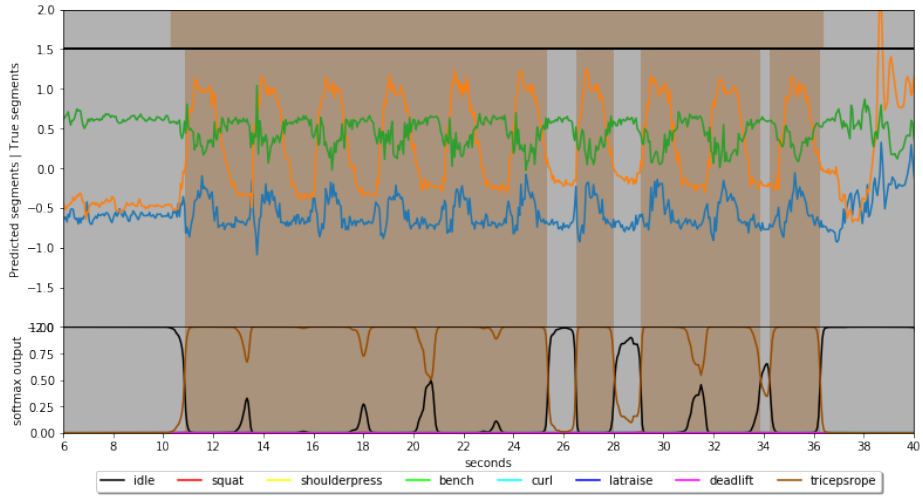


(a) Activity segment.

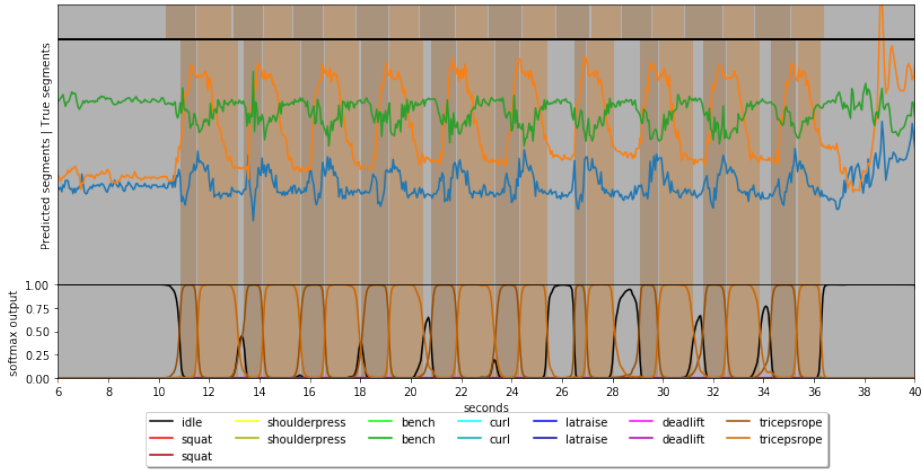


(b) Repetition segments.

Figure 5.6: Predicted activity and repetition segments for one of the sets of curl by participant 4. The model was trained with this person in the validation set. The colored segments above the top black line are the true segments. At the bottom is the softmax output, with illustrated colored segments based on the max value. sensor data is plotted for reference. Best viewed in color.



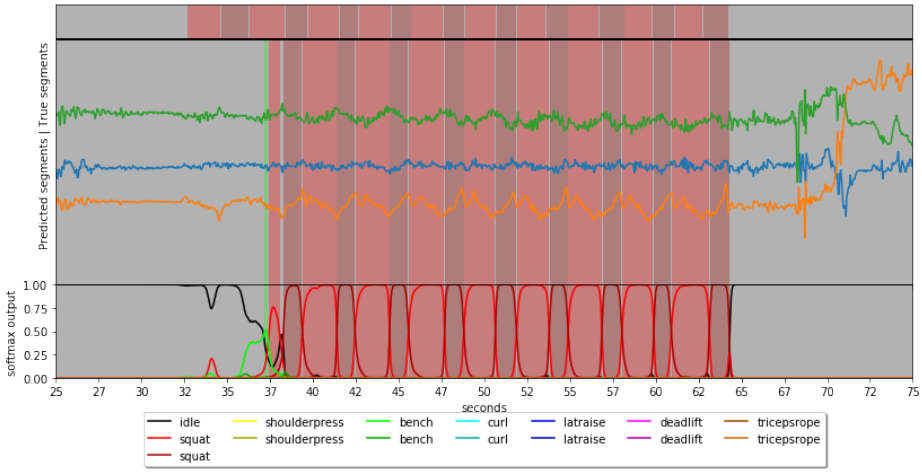
(a) Activity segment.



(b) Repetition segments.

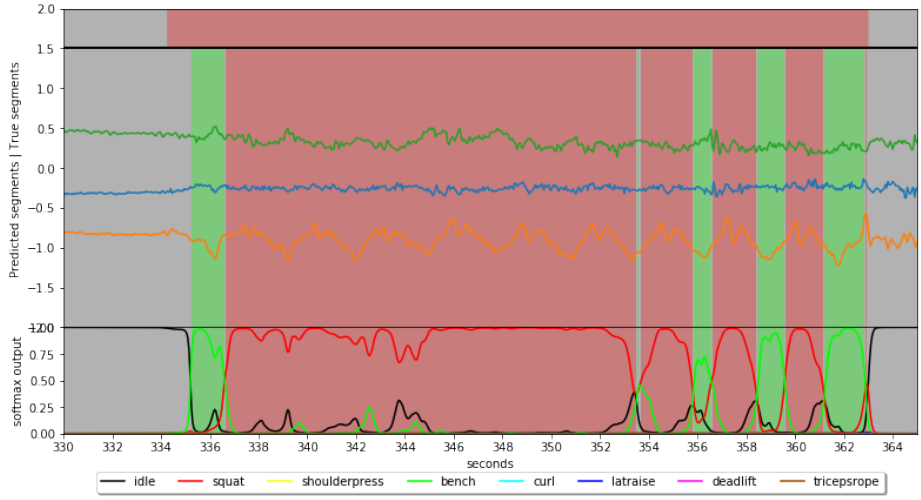
Figure 5.7: Predicted activity and repetition segments for one of the sets of triceps rope pushdown by participant 4. The model was trained with this person in the validation set. The colored segment above the top black line is the true segments. At the bottom is the softmax output, with illustrated colored segments based on the max value. sensor data is plotted for reference. Best viewed in color.

(a) Activity segment.

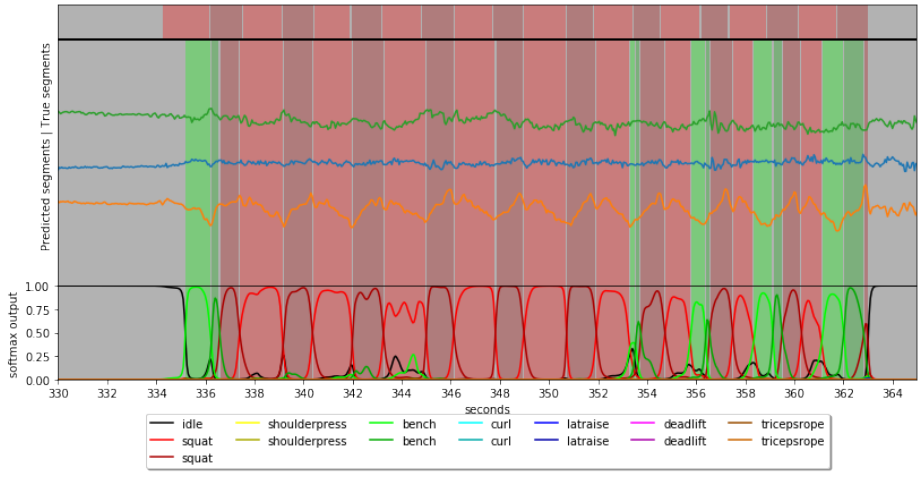


(b) Repetition segments.

Figure 5.8: Predicted activity and repetition segments for one of the sets of squat by participant 7. The model was trained with this person in the validation set. The colored segments above the top black line are the true segments. At the bottom is the softmax output, with illustrated colored segments based on the max value. sensor data is plotted for reference. Best viewed in color.

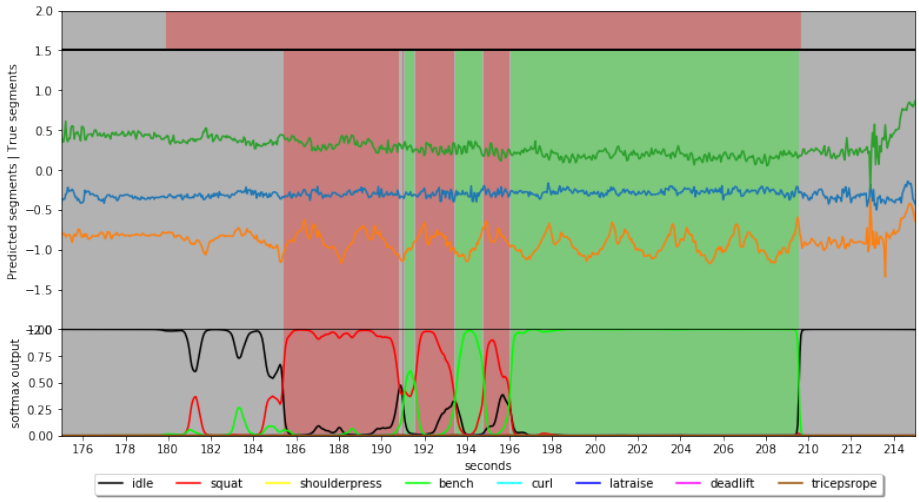


(a) Activity segment.

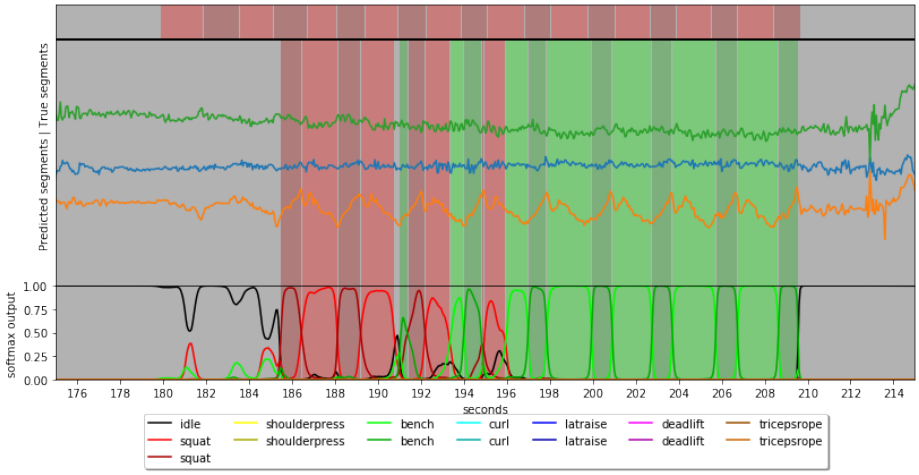


(b) Repetition segments.

Figure 5.9: Predicted activity and repetition segments for one of the sets of squat by participant 7. The model was trained with this person in the validation set. The colored segments above the top black line are the true segments. At the bottom is the softmax output, with illustrated colored segments based on the max value. sensor data is plotted for reference. Best viewed in color.

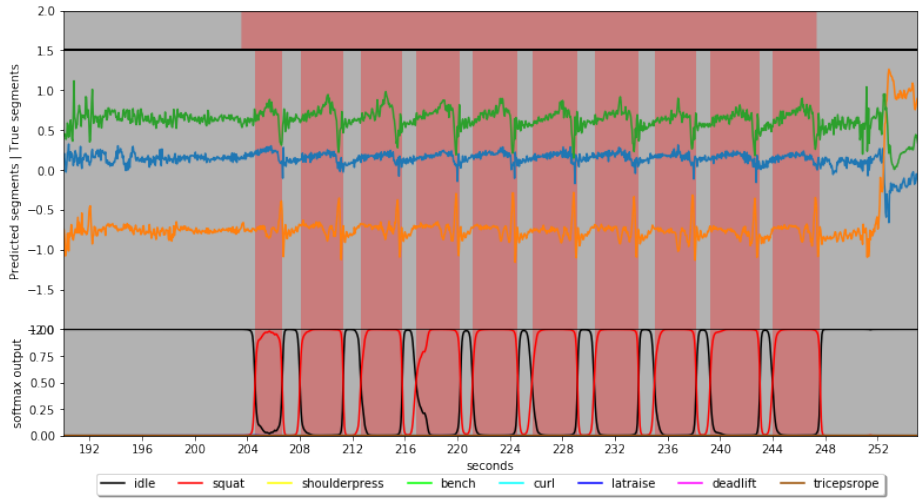


(a) Activity segment.

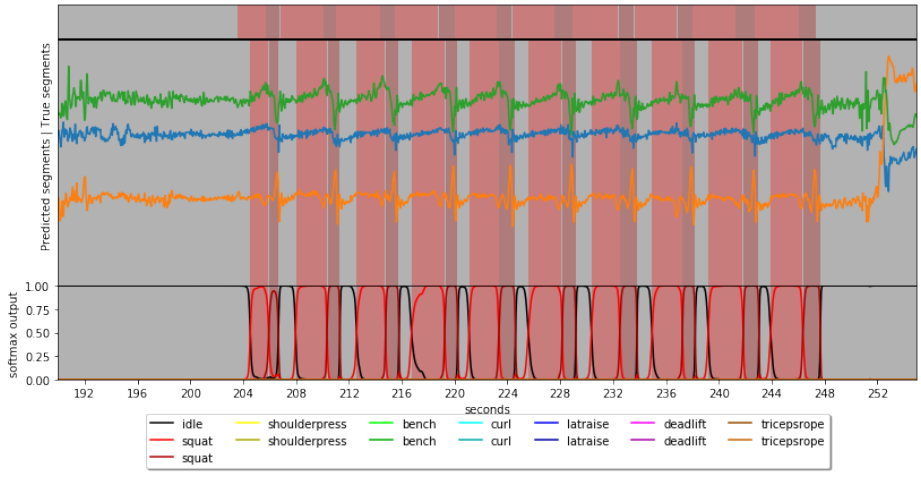


(b) Repetition segments.

Figure 5.10: Predicted activity and repetition segments for one of the sets of squat by participant 7. The model was trained with this person in the validation set. The colored segments above the top black line are the true segments. At the bottom is the softmax output, with illustrated colored segments based on the max value. sensor data is plotted for reference. Best viewed in color.

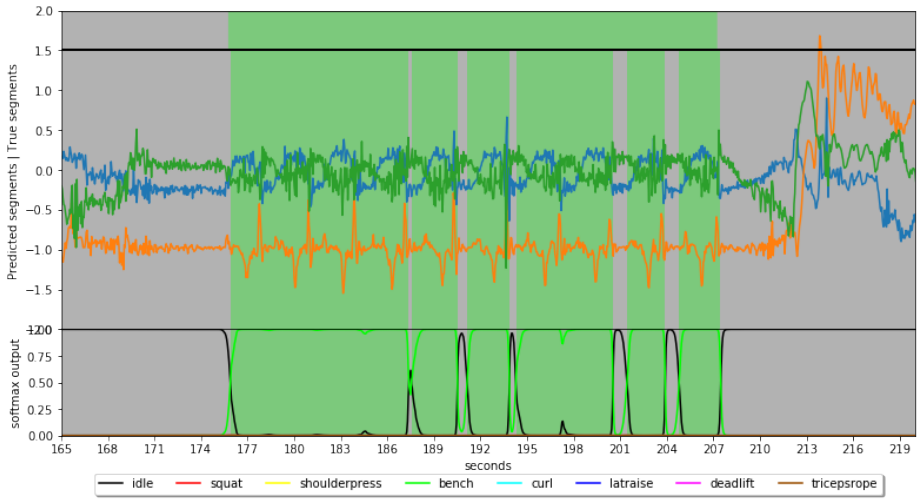


(a) Activity segment.

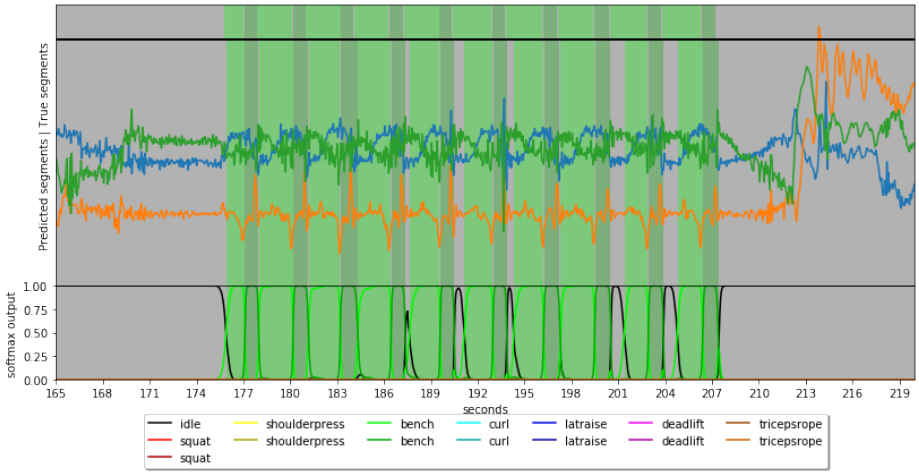


(b) Repetition segments.

Figure 5.11: Predicted activity and repetition segments for one of the sets of squat by participant 11. The model was trained with this person in the validation set. The colored segments above the top back line are the true segments. At the bottom is the softmax output, with illustrated colored segments based on the max value. sensor data is plotted for reference. Best viewed in color.

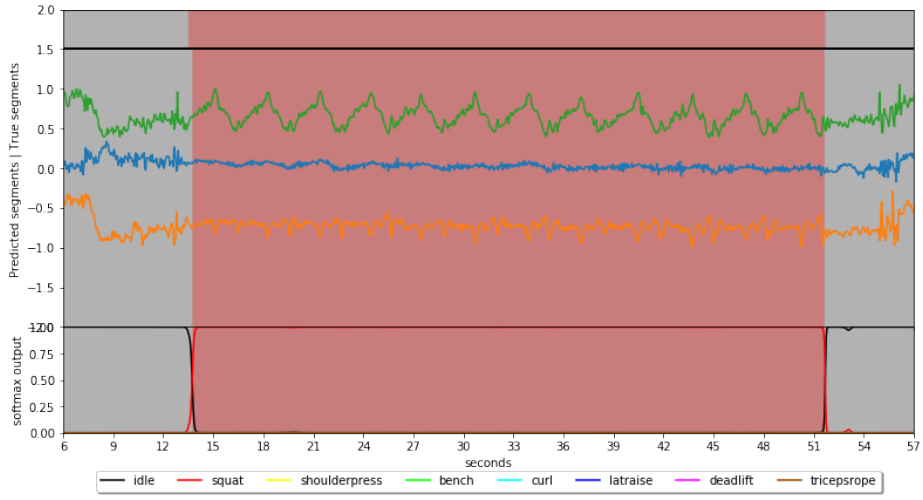


(a) Activity segment.

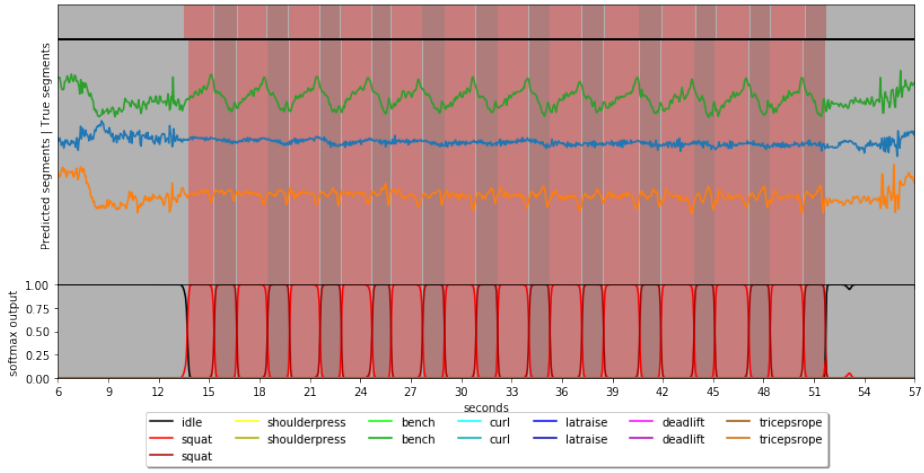


(b) Repetition segments.

Figure 5.12: Predicted activity and repetition segments for one of the sets of bench by participant 11. The model was trained with this person in the validation set. The colored segments above the top black line are the true segments. At the bottom is the softmax output, with illustrated colored segments based on the max value. sensor data is plotted for reference. Best viewed in color.



(a) Activity segment.



(b) Repetition segments.

Figure 5.13: Predicted activity and repetition segments for one of the sets of squat by participant 2. The model was trained with this person in the validation set. The colored segments above the top black line are the true segments. At the bottom is the softmax output, with illustrated colored segments based on the max value. sensor data is plotted for reference. Best viewed in color.

observing there were few cases where the predicted category fluctuated between the NULL class and an exercise during an actual set.

The worst accuracy for activity recognition was obtained when participant 4, 7 and 11 was in the validation set. The confusion matrix for the activity recognition and repetition recognition for these three participants are shown in Figure 5.3 for participant 4, Figure 5.4 for participant 7 and Figure 5.5 for participant 11. For participant 4, 7.7 % of the segments for activity category 4 (biceps curl) was predicted to be activity category 7 (triceps rope pushdown). For participant 7, 22.7% of the segments of activity category 1 (squats) was falsely predicted to be activity category 3 (bench press). It was observed during preliminary experiments with fewer participants that sometimes the exercises shoulder press, bench press and squats would be confused with each other. The reason for this is probably that the orientation of the left forearm is similar for all these exercises, and they all have a similar vertical motion when observing the data.

The accuracies for repetition recognition in Fig 5.1 are lower than for activity recognition, with an average accuracy of 88.5%, but this is to be expected. For repetition recognition, the primary source of error was not misclassifying segments of exercises as the NULL category, but misclassifications between the two states that comprise a repetition for a given exercise. This can be observed in the average of confusion matrices for all participants when in the validation set in Figure 5.1b. This comes from the fact that the model is not able to accurately predict the start and end of the two phases correctly. There are approximately ten times as many switches between these repetition states as there are between exercise segments and the NULL category for each set, as participants performed 8-12 repetitions each set. In addition, the error introduced when trying to capture timestamps for labeling repetitions with the stopwatch is more significant than for the labels created for the activity recognition even though the same timestamps are used to label the segments. This will cause errors in learning, and when evaluating, since the error data collection error varied between repetitions and participants. Since repetition phases last approximately 1.5 seconds, while an activity lasts about 30 seconds, a 0.1 seconds mismatch between the collected timestamps and events in data will cause 6.67 % error for each repetition phase while only 0.3% error for the activity segment.

For the participants with the worst obtained accuracies visualizations of predicted segments are provided to investigate what specifically went wrong. For participant 4, observe from Figure 5.6 how the model starts predicting segments of triceps rope pushdown two places in the middle of the set of curl. The confusion is understandable as the two movements are similar to in nature. The confusion between two exercises also happened for some of the squat sets of participant 7, shown in Figure 5.9 and 5.10. Here the squat exercise is confused with the bench press exercise. Pay attention to the data plots obtained in Figure 5.8a, 5.9 and

5.10 compared to one of the sets of squat of participant 2 in Figure 5.13, where the accuracy was near perfect. For participant 7, the pattern in the movements is mainly in the y-direction (orange data plot), while for participant 2, the main pattern is found in the z-direction (green data plot). The y-direction is the vertical axis, and the z-direction is one of the horizontal axes. This means participant 7 had a vertical orientation of the hand throughout most of the execution of the exercise. Participant 2 had a hand placement more horizontal to the ground, making the movement create the pattern in the z-direction. With a close grip on the bar during squat and wide foot stance, it will be natural to have a vertical hand orientation and movement. With a wide grip and narrow foot stance, the arm will be angled more horizontally, and the upper body will also angle more horizontally during the movement. This might explain the confusion with the benchmarks exercise, which almost only has movement in the y-direction and is therefore predicted instead of squat like in Figure 5.10. In Figure 5.8 there is more movement in the z-direction by participant 7 in the squat exercise, and it seems that the model is able to predict both the exercise type and repetition phases correctly accurately. Note however that even though the wrong exercise is predicted in Figure 5.6 and 5.10, the correct changed between repetition phases are predicted, although for the wrong exercise. This mainly supports the claim that the model can detect the two repetition patterns, and the execution of the squat exercise for participant 7 to be predicted as the bench press exercise.

Another source of error is that the exercise is not detected altogether, like the start of the set in Figure 5.10. Note that the first one and a half repetition in the set has data that is not very distinctive like the other repetitions in the set. This shows that the movement perhaps was to slow for the network to believe it was an exercise. Observe the softmax outputs, which does indeed immediately start to change when the set begins but does not become high enough for the exercise to be predicted. The fact that the model seemed to be confused about which exercise is being performed may also have contributed to not detecting an exercise.

The last type of errors found for the participants with bad results are those visualized in Figure 5.6, 5.11 and 5.12. Especially in 5.11, the model mispredicts the NULL category as a part of the first repetition phase, for every repetition throughout the set. Observe again the data for both the figures and the segments predicted as the NULL category. The data plots in these segments show close to no movement for these segments. Since this happens after every repetition is finished, it probably is because the participant took a little break between each repetition. This can also be observed by the difference in length of the two repetition phases in the true segments. The timestamps during data collection were created after each repetition phase is finished. If the participant waits after finishing repetition phase 2 for each repetition before starting repetition phase 1

again, this extra time will be given to the segment of repetition phase 1.

This could possibly be solved by letting the model predict with more delay than 1 second. This would let the model observe that a new repetition is coming, and therefore the previously observed data should be of repetition phase 1 instead of the NULL category. A more basic flaw is perhaps labeling this whole time segments as a repetition phase altogether. A correct approach would perhaps be to label this as a break between repetitions, and only label the actual up and down movement as the two repetition phases. This should probably be considered for future work when labeling repetitive events with longer and that have variable length breaks between them. For weightlifting with heavier weights, it is to be expected this break will occur, especially late in the set as the participant becomes tired. The participates in this study was asked to lift a weight they would control and might have performed the exercises more in an ideal way compared to how they would otherwise.

Lastly, it should be pointed out that even for the participants and exercises with the worst results, except for the confusion between exercises for the sets of squats performed by participant 7, the segmentation of repetition phases is possible to use. If segments of the NULL category inside sets, like in Figure 5.11, are removed the repetition phases are distinct and continuous. Filtering the outputs in this way, a straightforward algorithm could be used to count repetitions as transition changed between the two repetition phases categories. For participants and exercises with reasonable accuracy, which was the majority of the results, as shown in Figure 5.1, very little filtering should be needed to use this simple algorithm. One of the better which is shown in Figure 5.13 would require no filtering of the output to apply such a counting algorithm.

The results show that an efficient recurrent model integrating both activity and repetition segmentation provides useful results. The model is capable of generalizing to new persons, and the flaws in the results are mainly believed to be from the dataset consisting of few participants, and errors in labeling the collected data sequences.

Chapter 6

Conclusion

Section 6.1 discusses the work and results obtained in this project, as well as what could have been done differently to achieve better results. Since planning, data collection, data preparation, model building and the experiments have been performed over the course of this project, multiple ideas have been gathered on possible ways to improve and expand the project. These ideas are presented in Section 6.2.

6.1 Discussion

The method used to collect data might have caused significant errors in segmenting the two phases of each repetition. First off, the data stream and the stopwatch used to segment the data was started separately on two devices, although attempted to be performed at the same time. This might have caused a slight shift, but should not be very significant. A significant error was introduced to how the repetitions were labeled. It was difficult for the assistant to time the turning-points of each repetition as sometimes the movement transitions are smooth, or human error caused the assistant to press the stopwatch too early or too late. Since each phase of repetitions is around 1-1.5 seconds, a slightly missed timing on the stopwatch might have caused significant errors in the label data. Also, breaks between each repetition in a set were labeled as one of the two repetition phases. For one participant, this caused significant errors in one of the exercises when comparing the predictions to how the segments were labeled. One could argue the predictions were more correct then what was considered the true segments in this case, as breaks between each repetition in a set is not one of the repetition phases.

Manually trying to label the time series by looking at the graphs is also dif-

ficult and takes a long time. A better method would probably be to capture video of the participants performing the workout, sync this up with the events in the collected data, and register the time of the different events when watching the video. This would remove the human error in collecting the labels, as well as provide very accurate segments as one could obtain frame perfect accuracy. It would also be possible for participants to perform the entire workout by themselves without the need for an assistant because they could wear the data collection device and capture a video of themselves with a smartphone.

The performance of the model would probably be better with a dataset consisting of more participants. Repetition data from the same participant is highly correlated, and only 13 participants were collected in the dataset. The leave-one-subject-out experiment showed that some exercises were difficult to recognize for some of the participants. In particular, these exercises was squats and triceps rope pushdown, and for one participant, the biceps curl exercise. The errors where the NULL category was mispredicted inside a was argued to come from small breaks participants took between sets. Giving the model a larger look-ahead window when making predictions might have helped remove this from the model outputs. Cases, where two exercises were confused with each other, was discussed as might have come from the fact that the sensor only collected data from the participants left hand, and for some of the exercises, the execution may produce very similar data series. This similarity in data was observed in one of the participants where the orientation of the hand and execution of the squat exercise almost only produced a pattern in the data in the vertical axis, making it similar to the movement of a bench press exercise.

The breaks had some small segments of predictions that were believed to be one or two repetitions of an exercise. In reality, these were movements like bending down to pick up a water battle or weights or flinging arms around. The model was sensitive to these events, indicating that it strongly emphasized recent events. There are probably possible explanations that might cause this. One of them is that such single events are somewhat rare in the data. Therefore, the model had little incentive to learn the complexity of categorizing such events as the NULL category. Another reason might be that the model only had a context of 1 second of data ahead of the data point it is to classify. Since 1 second is not enough time to observe if multiple repetitions are performed, the network has to predict the event as an exercise, or it will need to discard at least the first part of each performed set to prevent these one-time events from being categorized as an exercise. This could show that one second of lookahead context might be too small for such a model, and better performance could be obtained with a larger window of context. For the solution used in this project, this might be troublesome, as it uses only the ability of an LSTM to remember, and a longer window of context means the network needs to incorporate new information while

still having to keep the memory of what happened many seconds ago. A possible solution to combat this would be to introduce the attention mechanism. Although this significantly increases the computational complexity, it would be able to use the built state of the LSTM from multiple positions in the sequence to determine a category.

During this project, data was also collected with a full body suit with a total of 16 sensors recording data from an accelerometer and gyroscope in 100 Hz. The sensors were placed at the forearms, upper arms, shoulders, hips thighs, shins, feet, the stomach, and the chest. The suit had both accelerometer and gyroscope sensors. This suit consisted of proprietary hardware and software, and a bug was discovered late in the project. The bug was caused by a sampling rate error, causing the recorded data to shift in time relative to the time elapsed increasingly with the length of the sequence. Since the timestamps were collected using another device, the events in the data from the suit and the registered timestamps did not match. To use this data, the timestamps would have to be manually edited, which time did not allow for the remaining time of the project.

By using deep learning to transform raw sensor data into predefined categories, it was possible to segment both exercise activities and segment repetitions into two phases, which can be further used to perform repetition counting. The advantage of this method is not having to do feature engineering for every type of exercise in order to count the repetitions performed during a set. This will provide a method that scales should the types of sensors used to capture data change, or with the inclusion of multiple sensors or sensors placed differently. In addition, it was shown that this was possible with a small single recurrent model solving both tasks simultaneously with only 1-second look ahead in the data series. This proves it should be possible to perform this task on end devices like smart-watches and phones in a power-efficient manner.

6.2 Future Work

The most obvious thing that could be done to extend this work is to collect a larger dataset and create a procedure and tool to effectively and accurately label the collected sequences. As discussed previously, a solution with videos of participants performing the workout would seem like an effective way to do this. Not only would this give more quality in the labeled data, but it might also make participants more comfortable performing the workout in the way that they usually do. Since the goal is to create tools that could automatically log a workout, the data should be collected as close to a real-life scenario as possible. Such a solution would allow this.

In the extension of the dataset, it would also be useful to include multiple worn sensors. Although the most user-friendly scenario would be only to wear

a single device like a smartwatch, one or two sensors on the upper arm and a foot would have the ability to capture nuances for a much more extensive set of exercises. Since deep learning is used for feature extraction and classification for both activity recognition and repetition recognition, the inclusion of multiple sensors would not cause significant required changes for the model to incorporate these new data streams, or to add additional exercises for that matter. Multiple sensors might have given better results as was the case in Soro et al. [2019]. Their exercises had more full body movements and therefore had the benefit of using additional sensors attached to one of the participant's feet. Perhaps this setup could help the model easier to differentiate squats from bench press and shoulder press.

The dataset could further be expanded by including more exercises. The exercises chosen in this project were selected because of their popularity and such that it was possible for participants to perform all of them during a single workout. Since only a single sensor was used, it was observed that three of the exercises tended to be confused with one another. How a more extensive set of exercises might influence this effect would be useful to investigate, as well as if the effect is solved by using multiple sensors at different locations on the body. It could also be investigated how including activities where the repetitive events are not of interest affects the results of the segmentation. Such activities might be walking, running, or cycling.

Once a larger dataset is obtained, it should be used to evaluate model architectures on a k-fold train, validation and test split, something that was not performed in this project. A limitation of the work in this project was the number of participants the data was collected from, and the variance among them. The dataset should be extended such that representative groups similar to the characteristics of all the participants should be possible to partition into the three splits.

Additional research can be done into the selection of model architecture and parameters. Also, for the given architectures, a hyperparameter search should be conducted to obtain the best possible results. A downfall of using only recurrent networks on the raw data is that the network might struggle on data with a high sample rate. The ability of the recurrent network to learn long term dependencies might lack the ability to remember or learn patterns in the time series, especially if the sample rate is high. A possible solution to solve this would be to use a small convolutional network on the raw data in order to extract features from nonoverlapping windows and feed these features to a LSTM. This would still maintain the efficiency of processing each data point in the time series only once and take advantage of convolutional layers ability to detect local patterns and the ability for recurrent networks to remember long term dependencies to segment the sequences. Doing this could prevent overflowing the recurrent networks with

the high-frequency inputs since the convolutional layers can be used to transform raw data with arbitrary sample rate into a frequency more feasible for a recurrent network to deal with.

For more application-driven research, it would be useful to investigate how to best filter or make use of the transformation of raw data sequences to categorical sequences in order to count repetition and transcribe workout sessions. Most exercises are performed with multiple repetitions and usually span more than 5 seconds. Using such rules, it would be possible to filter out false predictions of short one-time events during breaks that looks like a movement in an exercise. A strength of transforming the raw sensor data into categorized segments is that it makes it easy to apply domain knowledge in order to filter the output of the model. Therefore, general rules can be applied to categorized segments instead of trying to compose rules for each type of exercise in order to categorize activities or repetitions by counting peaks and valleys in the raw sensor data.

Bibliography

Das, D., Busetty, S. M., Bharti, V., and Hegde, P. K. (2017). Strength training: A fitness application for indoor based exercise recognition and comfort analysis. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1126–1129.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121, 2159.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs.LG]*.

Kofod-Petersen, A. (2015). How to do a structured literature review in computer science.

Maheedhar, M., Gaurav, A., Jilla, V., Tiwari, V. N., and Narayanan, R. (2016). Stayfit: A wearable application for gym based power training. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6290–6293.

Morris, D., Saponas, T. S., Guillory, A., and Kelner, I. (2014). Recofit: Using a wearable sensor to find, recognize, and count repetitive exercises. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 3225–3234, New York, NY, USA. ACM.

Qi, J., Yang, P., Hanneghan, M., Waraich, A., and Tang, S. (2018). A hybrid hierarchical framework for free weight exercise recognition and intensity

measurement with accelerometer and ecg data fusion. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 3800–3804.

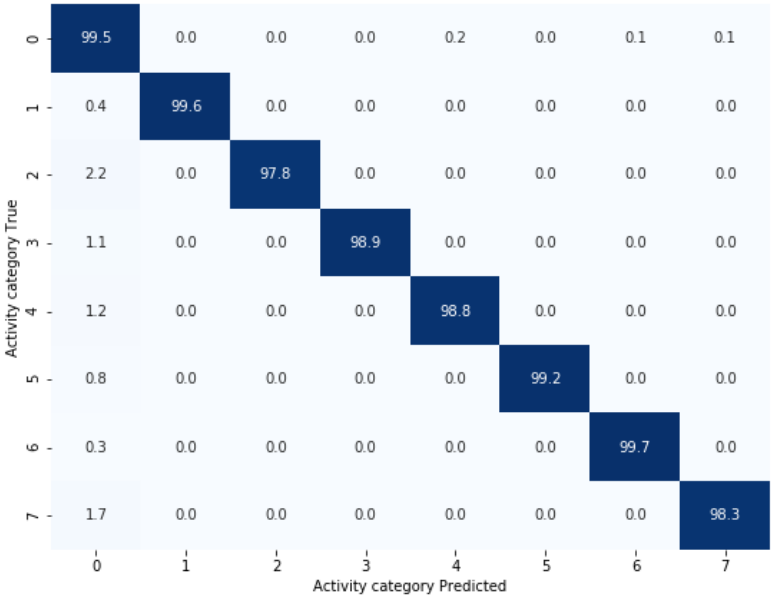
Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence - a modern approach, 3rd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall.

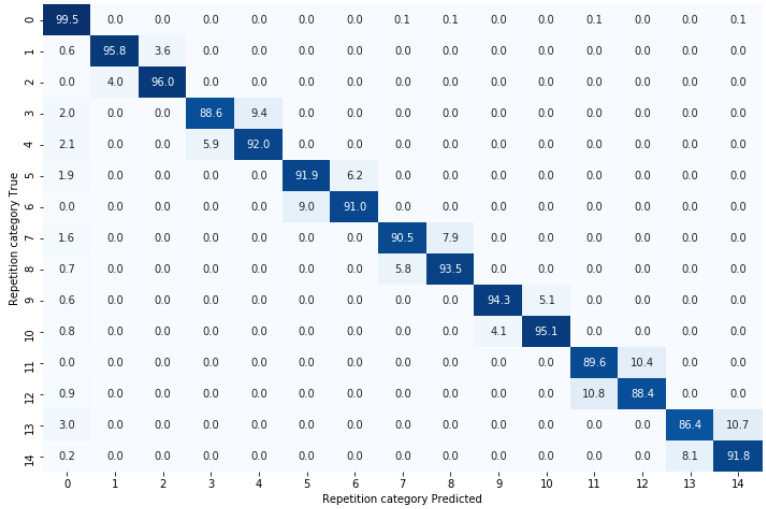
Shugang Zhang, Zhen Li, J. N. L. H. S. W. and Wei, Z. (2016). How to record the amount of exercise automatically? a general real-time recognition and counting approach for repetitive activities. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 831–834.

Soro, A., Brunner, G., Tanner, S., and Wattenhofer, R. (2019). Recognition and repetition counting for complex physical exercises with deep learning. *Sensors*, 19(3):714.

Appendices

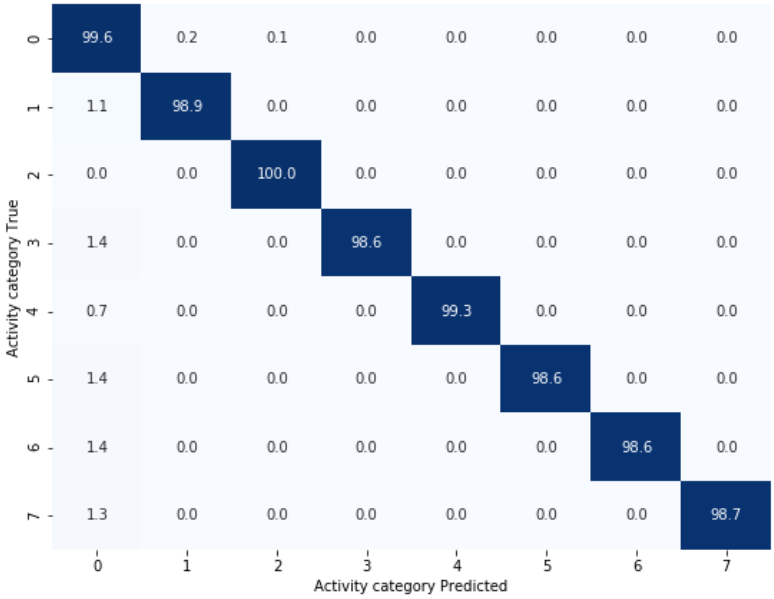


(a) Confusion matrix for the activity recognition.

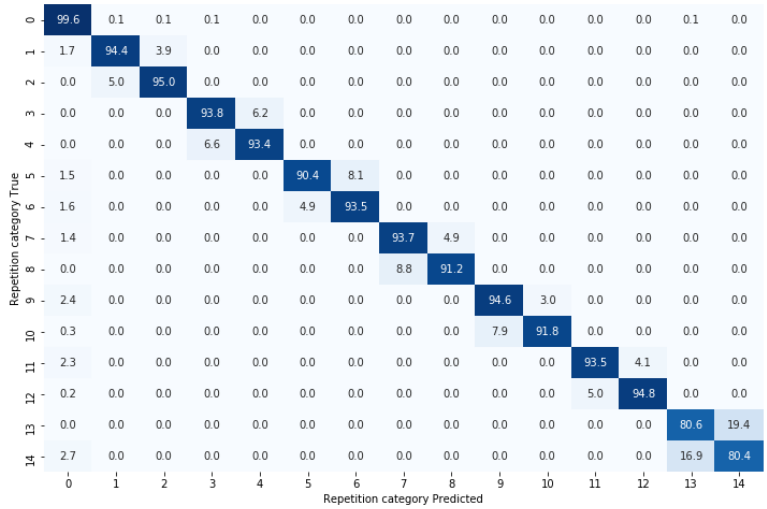


(b) Confusion matrix for the repetition recognition.

Figure 6.1: Confusion matrices for participant with id 0 when left in the validation set for the leave-one-subject-out experiment.

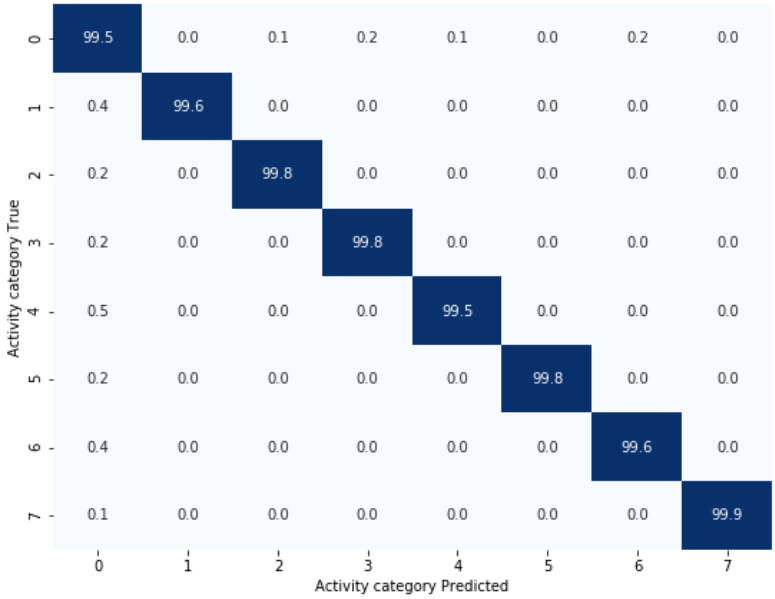


(a) Confusion matrix for the activity recognition.

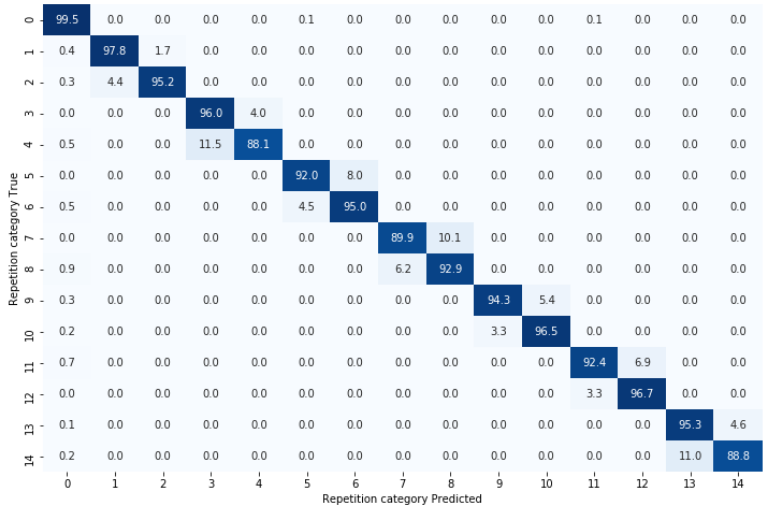


(b) Confusion matrix for the repetition recognition.

Figure 6.2: Confusion matrices for participant with id 1 when left in the validation set for the leave-one-subject-out experiment.

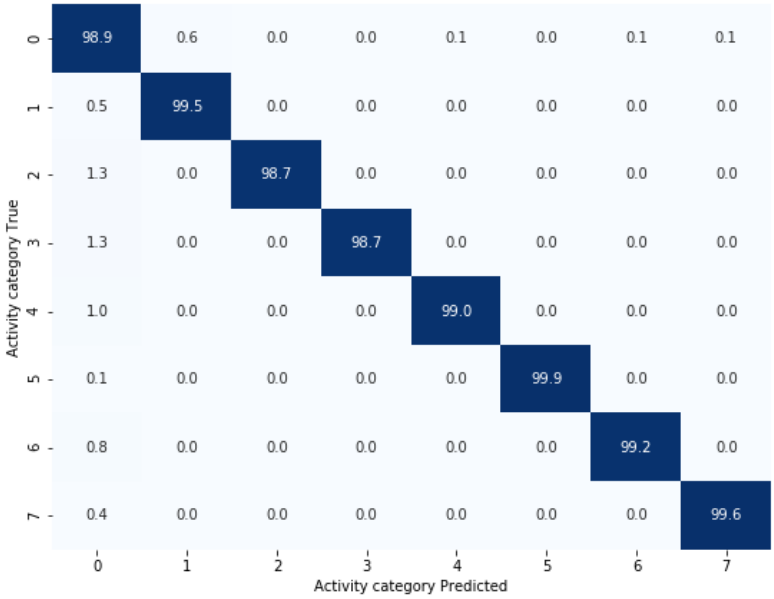


(a) Confusion matrix for the activity recognition.

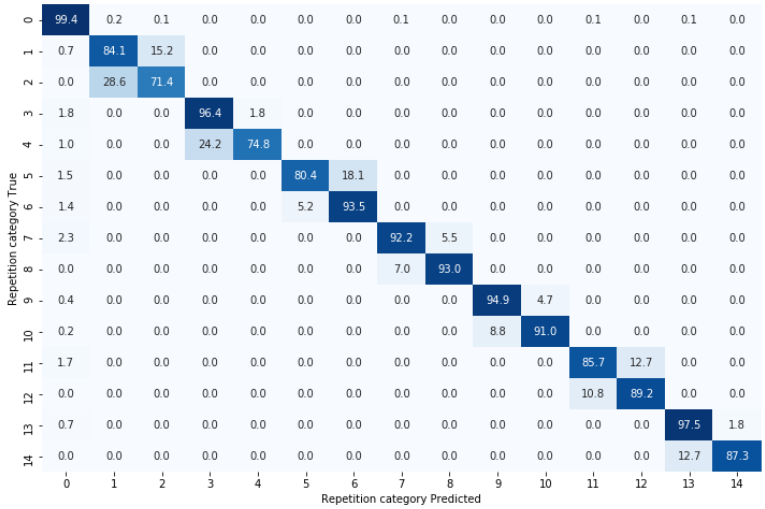


(b) Confusion matrix for the repetition recognition.

Figure 6.3: Confusion matrices for participant with id 2 when left in the validation set for the leave-one-subject-out experiment.



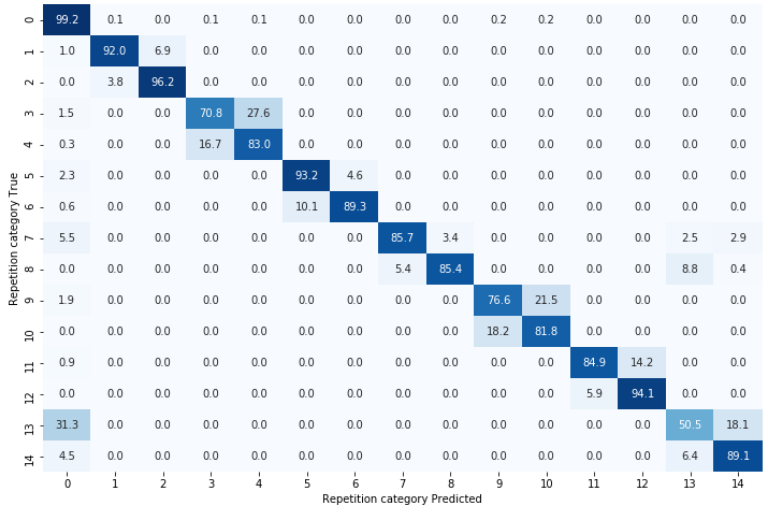
(a) Confusion matrix for the activity recognition.



(b) Confusion matrix for the repetition recognition.

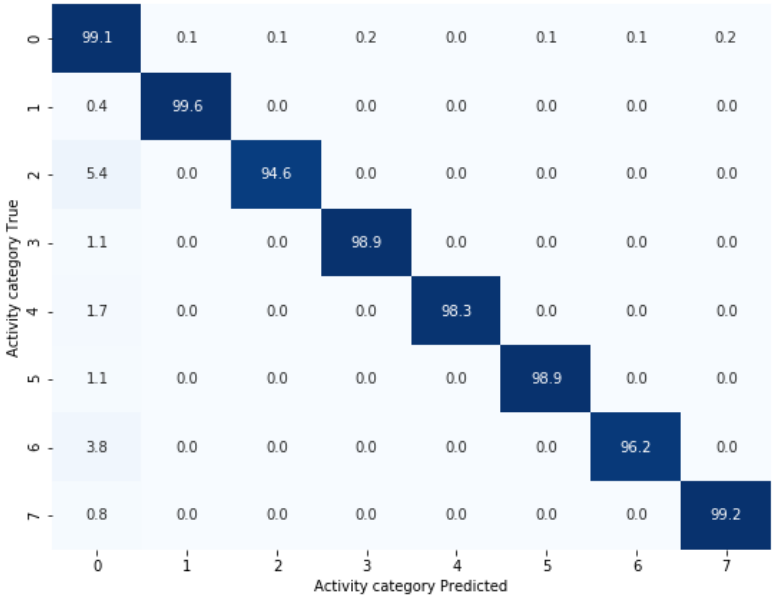
Figure 6.4: Confusion matrices for participant with id 3 when left in the validation set for the leave-one-subject-out experiment.

(a) Confusion matrix for the activity recognition.

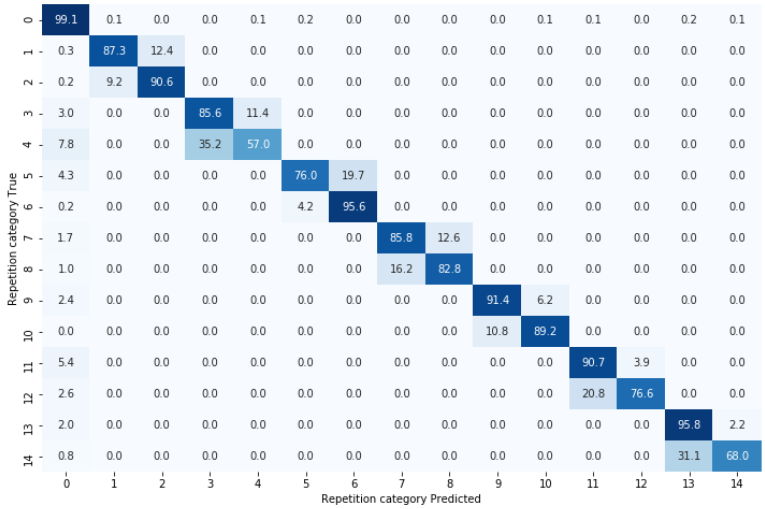


(b) Confusion matrix for the repetition recognition.

Figure 6.5: Confusion matrices for participant with id 4 when left in the validation set for the leave-one-subject-out experiment.

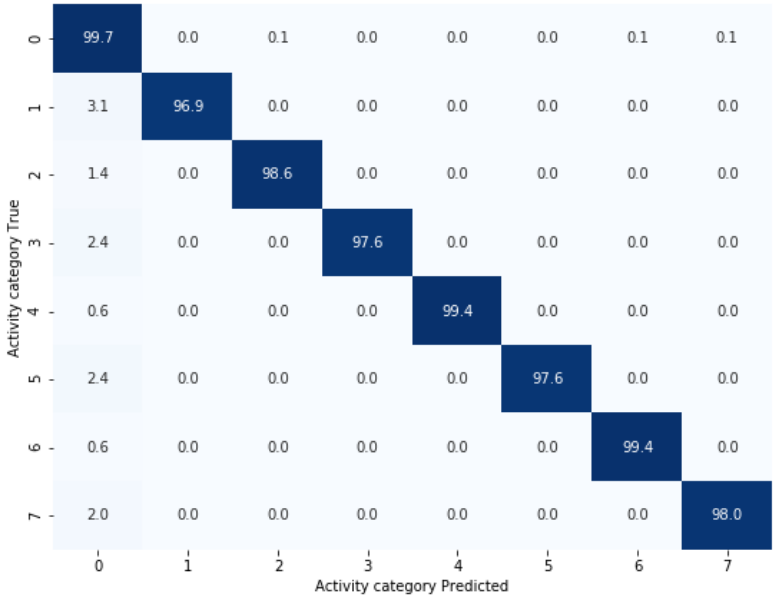


(a) Confusion matrix for the activity recognition.

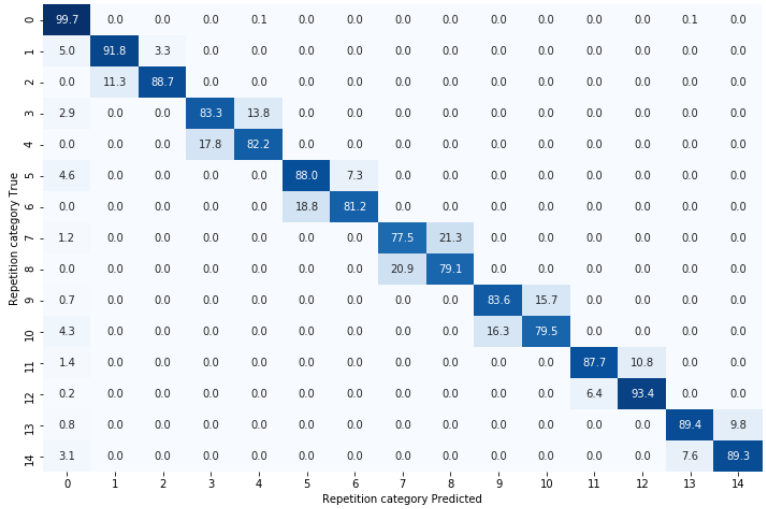


(b) Confusion matrix for the repetition recognition.

Figure 6.6: Confusion matrices for participant with id 5 when left in the validation set for the leave-one-subject-out experiment.

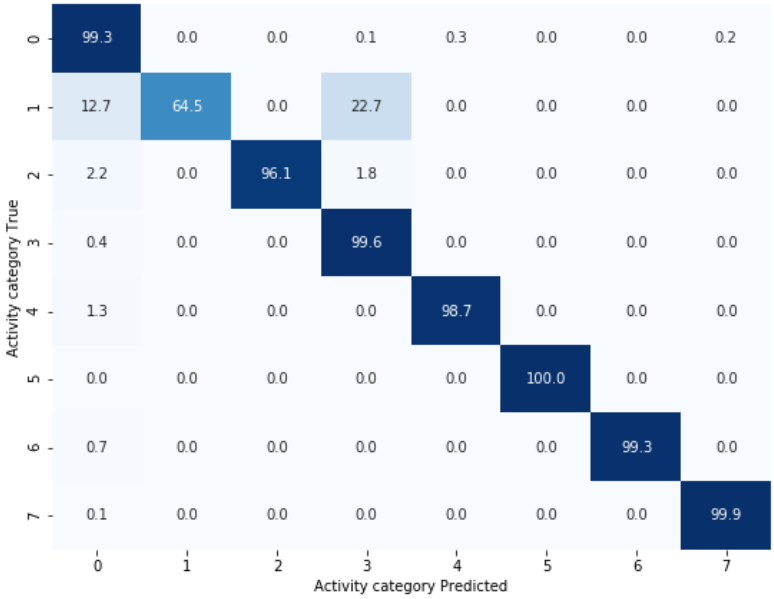


(a) Confusion matrix for the activity recognition.

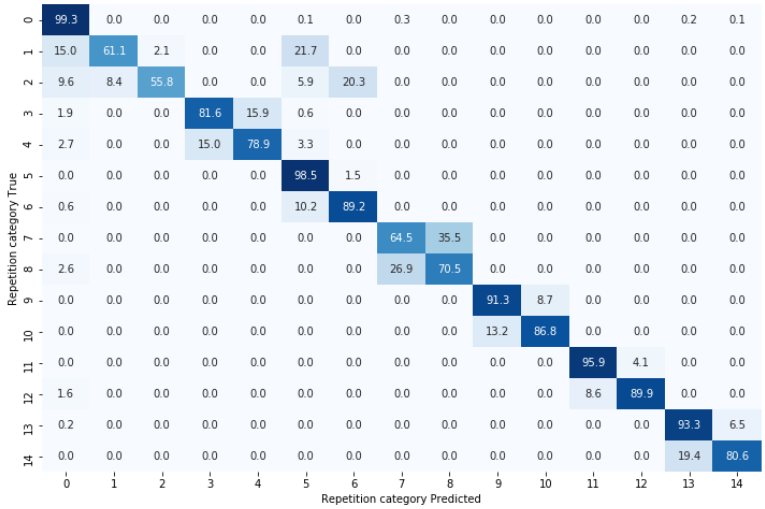


(b) Confusion matrix for the repetition recognition.

Figure 6.7: Confusion matrices for participant with id 6 when left in the validation set for the leave-one-subject-out experiment.

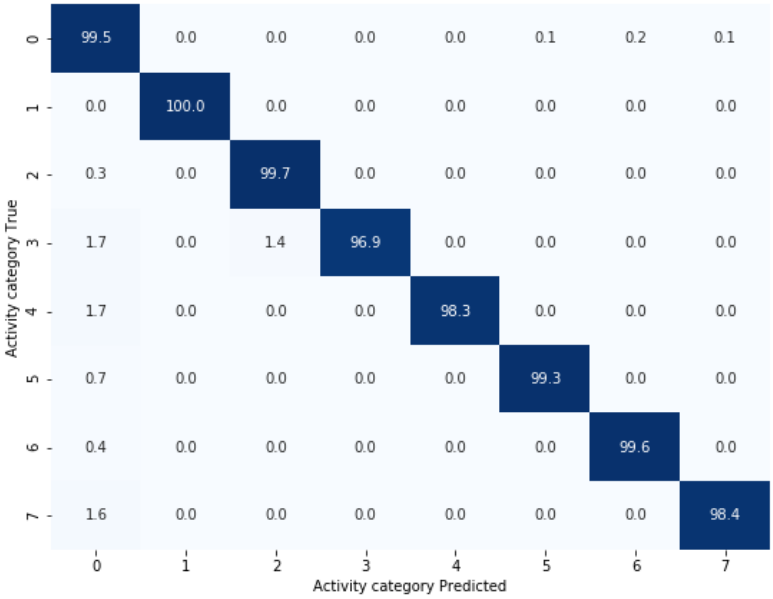


(a) Confusion matrix for the activity recognition.

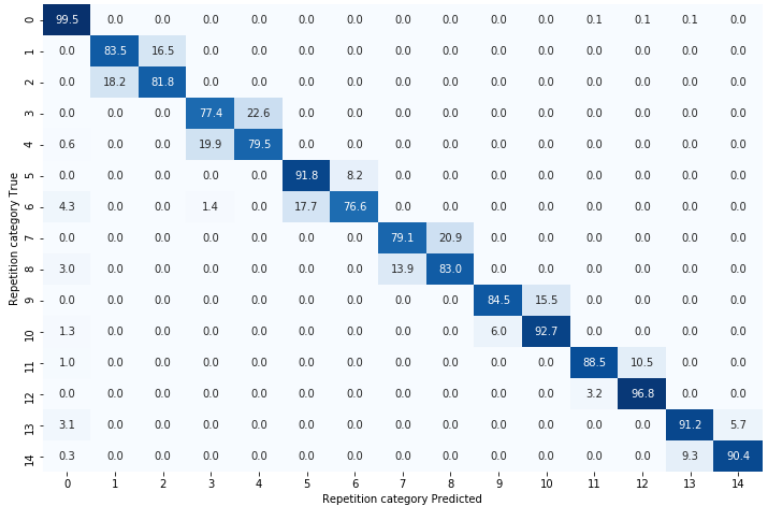


(b) Confusion matrix for the repetition recognition.

Figure 6.8: Confusion matrices for participant with id 7 when left in the validation set for the leave-one-subject-out experiment.

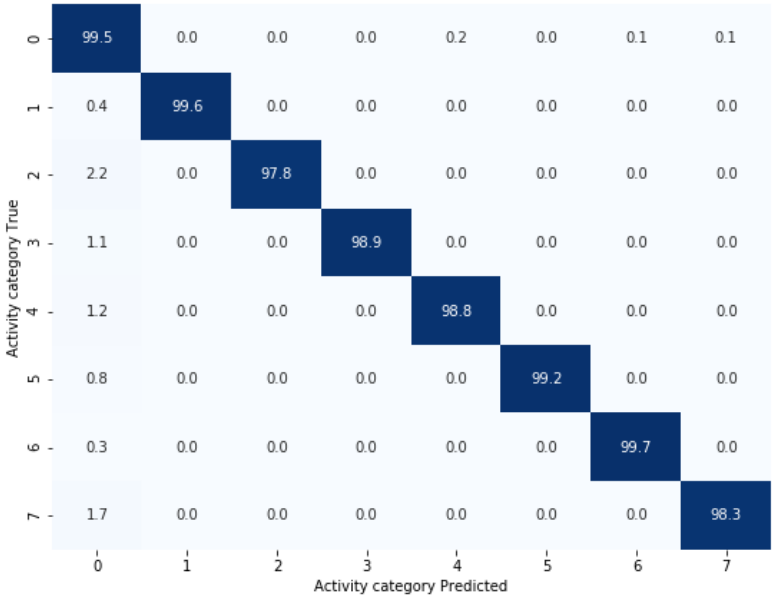


(a) Confusion matrix for the activity recognition.

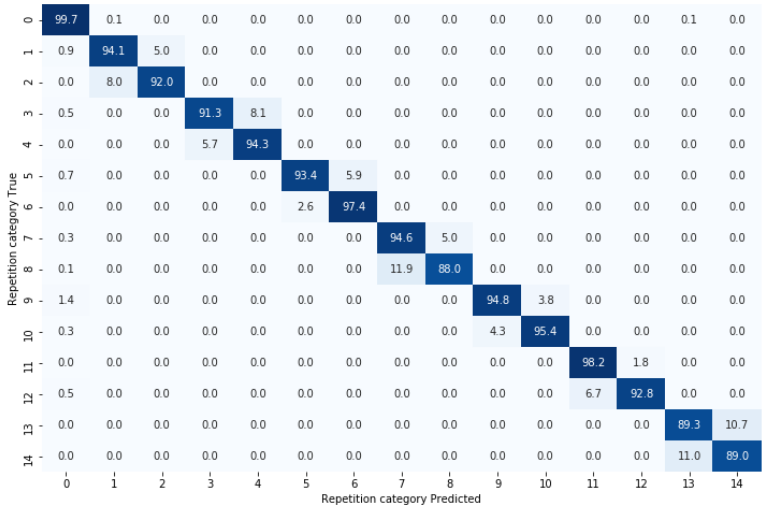


(b) Confusion matrix for the repetition recognition.

Figure 6.9: Confusion matrices for participant with id 8 when left in the validation set for the leave-one-subject-out experiment.

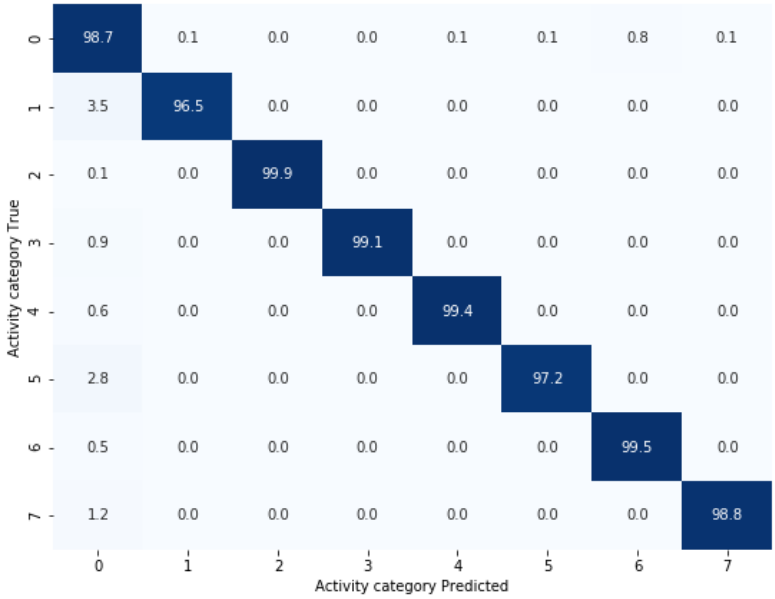


(a) Confusion matrix for the activity recognition.

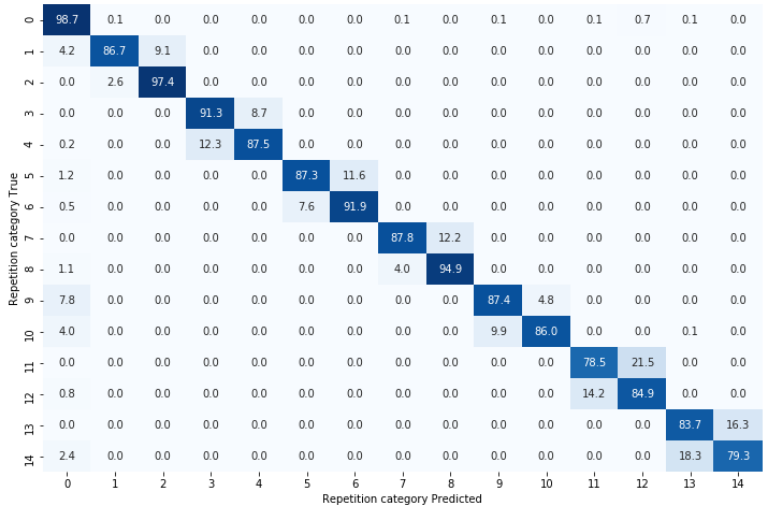


(b) Confusion matrix for the repetition recognition.

Figure 6.10: Confusion matrices for participant with id 9 when left in the validation set for the leave-one-subject-out experiment.

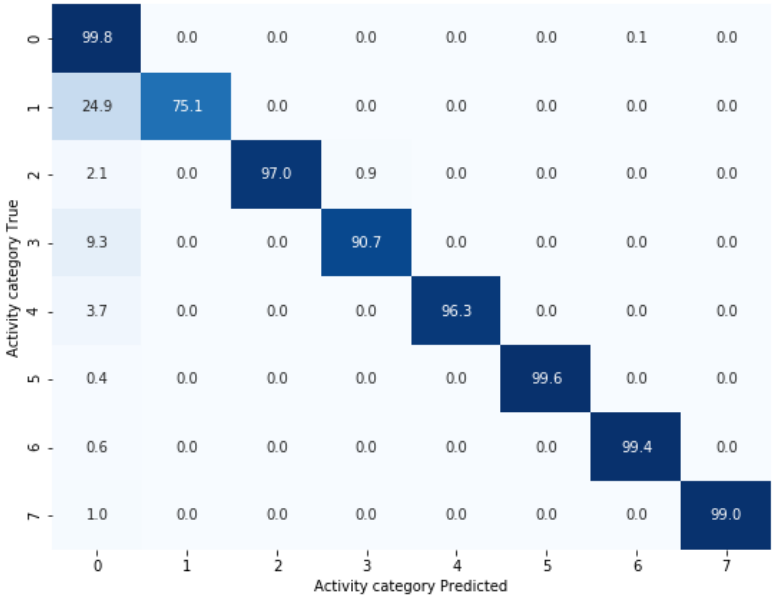


(a) Confusion matrix for the activity recognition.

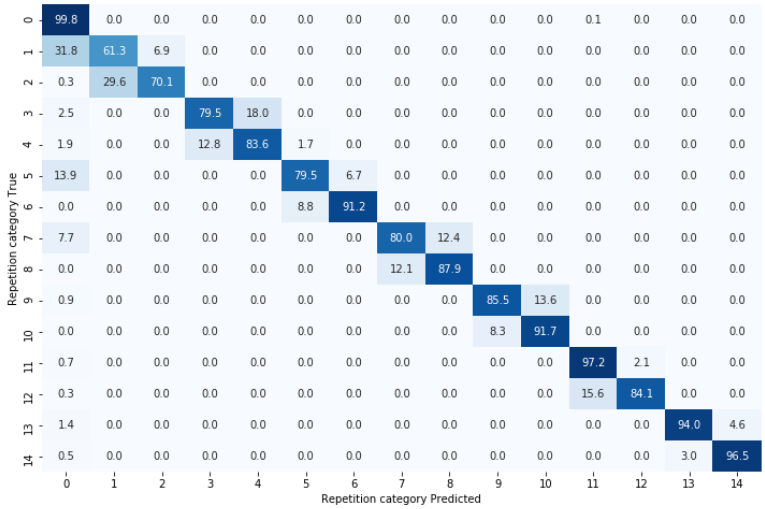


(b) Confusion matrix for the repetition recognition.

Figure 6.11: Confusion matrices for participant with id 10 when left in the validation set for the leave-one-subject-out experiment.

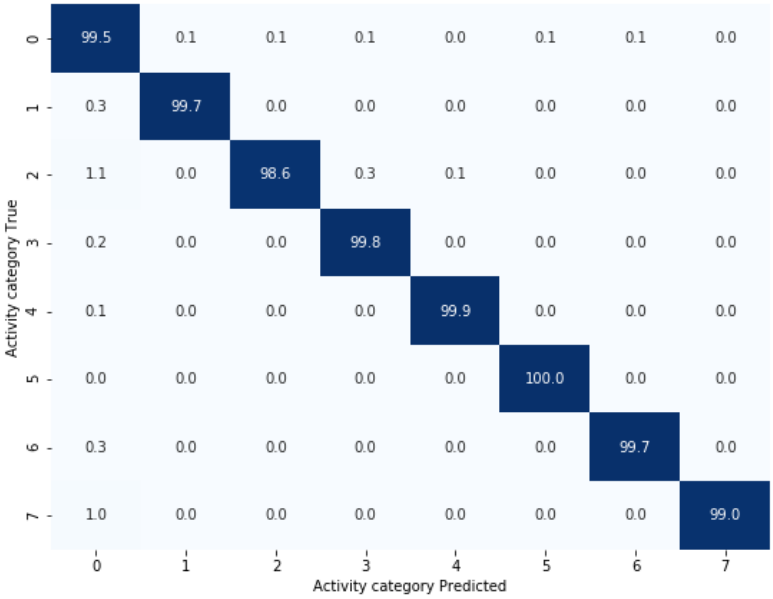


(a) Confusion matrix for the activity recognition.

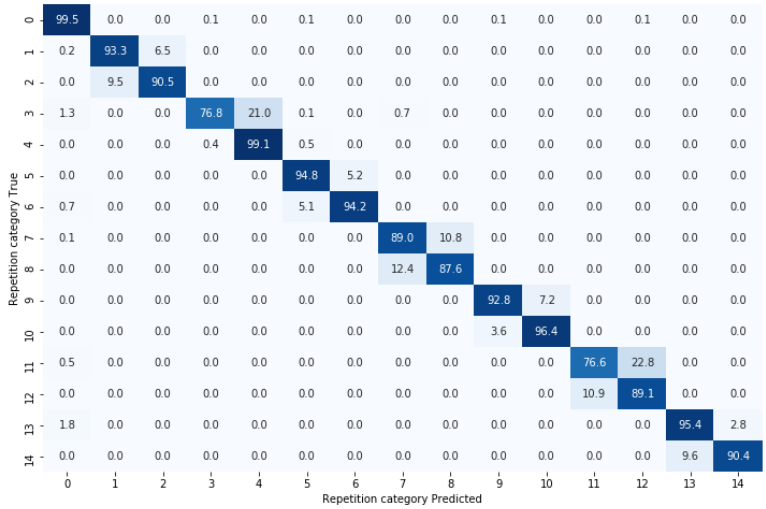


(b) Confusion matrix for the repetition recognition.

Figure 6.12: Confusion matrices for participant with id 11 when left in the validation set for the leave-one-subject-out experiment.



(a) Confusion matrix for the activity recognition.



(b) Confusion matrix for the repetition recognition.

Figure 6.13: Confusion matrices for participant with id 12 when left in validation set for the leave-one-subject-out experiment.