DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's thesis in Informatics: Games Engineering

Hand-Gesture Classification via Neural Networks: Improving Accuracy with Continuous Use

Cedric Fromm

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's thesis in Informatics: Games Engineering

Hand-Gesture Classification via Neural Networks: Improving Accuracy with Continuous Use

Handgesten Klassifizierung mithilfe von Neuronalen Netzwerken: Verbesserung der Genauigkeit durch kontinuierliche Nutzung

Author:	Cedric Fromm
Supervisor:	Prof. Gudrun Klinker
Advisor:	M.Sc. Sandro Weber
Submission Date:	15.09.18

I confirm that this Bachelor's thesis in Informatics: Games Engineering is my own work, and I have documented all sources and material used.

Munich, 15.09.18 Cedric Fromm

Acknowledgments

I would like to thank Prof. Gudrun Klinker, Sandro Weber and everyone at the research group for augmented reality for accepting and helping me with my thesis. I also want to thank my family and my friends for always supporting me. Big props to my MangoHuntaz bois for being awesome and huge shoutouts to my dogs, love you all <3.

Abstract

Thanks to wearable technologies, hand gesture recognition tasks are presented with meaningful data due to the use of accelerometers, gyroscopes, and surface electromyographic sensors for example. In order to make use of the recorded data, classifiers such as artificial neural networks are used. Training these networks however, can often prove difficult and time consuming. The goal of this thesis is to develop, train and evaluate a recurrent neural network architecture for the purpose of classifying rock, paper, scissors gestures using surface electromyographic signals. The developed recurrent model is compared with an already established convolutional neural network architecture, which will also be trained and evaluated during this thesis. For this, a dataset of 6000 gestures is recorded by 15 different persons using the Myo armband. The developed recurrent neural network model turns out to achieve higher accuracy of 93.71% compared to the convolutional model's accuracy of 82.57% while also being less complex, thus being more memory efficient and faster to train.

Contents

1 Introduction

1.1 Motivation

Over the last decade, speech and gesture recognition have found their way into the daily life of many people thanks to smartphones and smartwatches, or products like the Amazon Echo or Fitbits. They all serve different purposes and roles, such as fitness tracking, health assistants, and orientation or convenience purposes, such as sending and receiving messages. For most of these applications, some kind of interaction between the user and the device is required. Traditional input devices like keyboards or touchscreens are unfeasible as many devices do not have an integrated display. In these cases, speech or gesture recognition is often used as it is a natural way of interaction between user and application without the use of a display, buttons or any kind of peripheral hardware. Moreover, on-body sensors and wearables can also be used to add speech and gesture recognition functionalities to existing applications, making them more accessible. This thesis will solely focus on gesture recognition.

In order to capitalize on the collected data of the sensory devices, gesture classification needs to be performed. Gesture classification is the task of classifying a recognized gesture into a set of predefined gestures. Sensors like accelerometers, gyroscopes or magnetometers produce streams of output data in different ranges, dimensions, and frequencies. In order to perform gesture classification, these data signals need to be processed in a way such that meaningful information about the performed movement can be inferred. In the past, stochastic methods, such as hidden Markov models were used for this purpose. These methods require meaningful features that have to be handcrafted and chosen carefully from the raw data. This means that a lot of empirical tests and expertise was needed to reach a promising classifying accuracy. Nowadays, a lot of this work has become obsolete due to the use of artificial neural networks. Neural network architectures are able to learn how to extract meaningful features from the data themselves.

Compared to single poses, gestures are dynamic, which means that they have a dependency on the time domain. In order to capture these dynamic properties, recurrent and convolutional neural networks can be used. Recurrent neural networks are able to capture the relation between the data points in the time domain thanks to forward edges in the network structure. This allows for a better accuracy when classifying dynamic data like gestures. Convolutional neural networks can also be used to classify sequential data by treating the data sequence as a two-dimensional image, where each row contains the input data of a single time step.

Creating and training a neural network is not trivial. Each application expects different properties and qualities from the classifier. Products like the Amazon Echo must work for any user without any initial training or calibration to make it usable out of the box. Other devices, like health assistants, are only used by a single user and require the highest

accuracy possible to avoid any error. A lot of research has been done using different neural network architectures for the task of speech and gesture recognition. The parameters chosen for the different network models are the results of various tests and research that tailor the model to the task at hand. When confronted with a recognition problem, it is often difficult to tell which architecture is the best to use. The choice depends on many different properties such as available data and time, sensors and computational power of the device, and maximum tolerated latency.

The purpose of this thesis is twofold: The first goal is to build a robust classifier for gesture recognition using surface electromyographic sensors (sEMG). The second purpose is to give an overview on the different qualities of two neural network architectures, namely recurrent and convolutional neural networks. For this, a gesture recognition task is presented. The task is to recognize and classify rock, paper and scissors gestures performed by multiple users. The Myo armband, a wearable device equipped with 8 sEMG sensors, is used as the input device. A training data set will be created by recording sample gestures from 15 volunteers. A recurrent and a convolutional neural network architecture will be developed and trained for the recognition task. Finally, the networks will be evaluated and compared using different metrics like accuracy, number of training steps required to reach a certain accuracy, or how well the model generalizes between multiple users.

1.2 Contributions

This thesis investigates the usefulness of recurrent and convolutional neural networks for the purpose of gesture classification using sEMG data. The recurrent neural network architecture is developed using the tensorflow API. The convolutional neural network architecture is taken from [Gol18]. A data set of 6000 gestures performed by 15 different people has been recorded using a rhythmic training environment developed by using the Unity Game Engine. The models are trained using the recorded dataset and their performance is then evaluated and compared. The results are summarized in Table 1.1. The definition of the metrics can be found in Section 3.4.

	Recurrent model	Convolutional model
Highest accuracy	0.9371	0.8257
How many steps until x% accuracy	21666 ($x = 90$)	80333 ($x = 80$)
Estimated number of weights	800.000	10.800.000
Computational latency	16.23ms	15.01ms
Prediction latency	1–3 time steps	1–3 time steps
Generalization	0.6791	0.6916
Specialization	0.9786	0.8964

Table 1.1: Evaluation results of both neural network models

The recurrent neural network model performed better for almost all metrics, making it the preferred choice for gesture classification using sEMG data.

1.3 Thesis Outline

In Chapter 2, the required background knowledge about gesture recognition, deep learning and the learning process of neural networks is introduced. Further, a brief explanation of the tools used for the implementation of the data recording environment and the neural networks is given.

Chapter 3 describes the problem setting tackled in this thesis. The data recording process and the resulting datasets are presented, and the metrics used to evaluate and compare the different neural network models are introduced and explained.

In Chapter 4, the development of the recurrent neural network architecture is shown. Parameter choices are justified by doing multiple tests and the final structure is presented and evaluated.

Chapter 5 explains the structure of the convolutional neural network. The chosen parameters are highlighted and the model is finally evaluated. A comparison between this model and the developed recurrent neural network is made in Chapter 6.

To conclude the thesis, a brief summary of the results as well as some final thoughts about possible future developments and related work is given.

2 Preliminaries

2.1 Gesture Recognition

2.1.1 Overview

Gestures refer to the orientation of the whole/part of the human physique for the purpose of conveying non-verbal information [Kon18]. Common examples of gestures are a thumbs up or a fist pump after a won sports match. Records of one of the oldest and most widely known gestures, the handshake, date way back in history all the way to 500 BC [Hum17]. When we speak about gesture recognition today, the most common usage is still to convey non-verbal information, but not between humans, rather between a human and a computer. The idea of using gestures as a way of controlling a computer dates back to 1983 [SS15]. Today, gesture recognition is widely used in different applications, like gaming with Microsoft's Kinect or taking remote photographs [Gon15]. Further research has been dedicated aiding handicapped persons, for example, gesture control for wheelchairs [Pan+14].

For computer applications, gesture recognition is defined as the task to recognize gestures and being able to understand and classify them. It is important to note that generally a distinction is made between pose and gesture recognition. The important difference between a pose and a gesture is that a gesture has a dependency on the time domain, or to phrase it simpler: it is dynamic. This means that a snapshot of a pose would likely contain enough information to detect and understand the depicted pose whereas a single snapshot of a gesture, for example waving your hand, would not be enough to fully recognize and classify the gesture. This means that instead of looking at single images, a sequence of connected images is used to perform gesture recognition. Before the advent of deep learning and other modern machine learning techniques, stochastic approaches, such as hidden Markov models or sliding window techniques were mostly used for gesture recognition. However, these approaches require handcrafted features, which require a lot of work and experience [OR16]. Today, deep learning and neural network architectures usually outperform traditional classifiers and are used for many cutting-edge technologies like face recognition [App17], recommender systems [Dia17] or autonomous driving [Dal18].

2.1.2 Hand Gesture Recognition

In order to recognize hand gestures, a variety of devices have been developed in the past. Notable examples are the Leap Motion, Unlimited Hand, the Myo wristband or the Nintendo Power Glove. These devices share a common goal, namely detecting hand movement and recognizing gestures, but the technological approaches differ. These approaches can be divided into two different groups: vision-based and sensor-based gesture recognition. This work focuses on sensor-based approaches.

Sensor-based approaches do not feature a camera but sensors like accelerometers, gyroscopes or sEMG sensors. These sensors are able to measure, for example, the relative change in motion, orientation or muscular activity. This data can be used to deduce the gesture performed between two time steps. The advantages of sensor-based approaches are that the sensors have a low energy consumption, are lightweight and do not require a wired connection, which makes them very mobile. Hence, sensor-based approaches are very popular for wearable technology.

2.1.3 Myo Wristband

Because of the small size of sensors, it is easy and convenient to install them in accessories, like watches or wristbands. For the gesture recognition task in this Bachelor thesis, the Myo armband is used to track all relevant information. The Myo armband is a wearable armband, made by Thalmic Labs [1]. It was released in 2014 and features not only a nine-axis inertial measurement unit including an accelerometer, a gyroscope, and a magnetometer but also, and most importantly, eight medical grade stainless steel sEMG sensors. sEMG is the abbreviation for surface electromyography and describes the medical diagnostic technique of measuring the electrical activity produced by muscles [Tak12]. To phrase it simpler: Thanks to eight conductive measuring units installed in the myo, the armband is capable of measuring muscle activity around the arm. This information can be used to detect hand gestures and poses done by the user. The software that comes with the armband is capable of distinguishing between five different poses:

- Fist: The user has his hand closed into a fist.
- Fingers spread: The user has spread its fingers apart
- Wave out: The user opens his flat hand with the palm facing away from him to the front, and the fingertips facing away from the body(To the right side, if it is the right hand and vice versa).
- Wave in: The same pose as wave out, but the fingertips now face the other sides respectively.
- Relaxed: No other poses are detected.

The motivation for developing new features for the Myo armband is manyfold:

- The classifier provided by Thalmic Labs cannot easily be extended to classify other poses or gestures, but it would be interesting to be able to create custom gestures for the classification.
- The standard Myo classification only distinguishes between poses and not gestures, but it would be interesting to see how well the Myo would be able to classify gestures, using neural networks.

[1] https://www.myo.com/

- The sEMG sensors are quite unique, and it would be interesting to know how well sEMG data can be used for hand gesture classification.

Even though the Myo armband comes with more sensors than just the sEMG sensors, this thesis will only focus on using the sEMG data for the classification task, as one of the goals is to find out how well the sEMG data can be used.

2.2 Deep Learning

There are two major approaches to artificial intelligence: logic-based and stochastic methods. The latter tries to acquire knowledge through pattern recognition. These kinds of approaches are called machine learning, and deep learning is one of many machine learning techniques. The basic idea of recognizing patterns starts with the search for features. A feature can be described as a piece of information regarding the object of interest or the pattern that needs to be recognized.

It is often difficult to find meaningful features for the problem at hand. For example, for facial recognition, the skin tone, eye and hair color, or the size and shape of the mouth all seem to be meaningful. But what about the distance between the eyes or the height of the ears? Furthermore, a great amount of work is needed to record and create a large data set including all the chosen features.

It is desirable to have an algorithm that takes an image, recognizes a face within the image, extracts all the relevant features itself and then uses the information to classify the face. Deep Learning is an approach which aims to do exactly that. This thesis focuses on two types of deep neural network architectures: the convolutional neural network and the recurrent neural network.

2.2.1 Recurrent Neural Networks

Recurrent neural networks, in contrast to conventional feed-forward neural networks, contain feedback loops. Intuitively, an input at time step t effects the input at time step $t + 1$. Mentioning time steps already reveals that a recurrent neural network stops looking at single images of data examples in isolation, but instead aims to classify a series of successive data points, such as a video stream or a sentence of natural language for example. The recurrent structure provides the network with the ability to build an internal temporal memory [Spi15], which can identify patterns in data sequences and use these for classification. Recurrent neural networks find a lot of usage in fields like natural language processing or gesture recognition for these kinds of reasons [You+18].

The general structure of a recurrent neural network is very similar to the structure of a multilayered perceptron. Generally recurrent neural networks feature only one hidden layer, which differs slightly from the general model, in that it contains feedback loops. This is visually represented by each hidden node linking to itself, as can be seen in Figure 2.1.

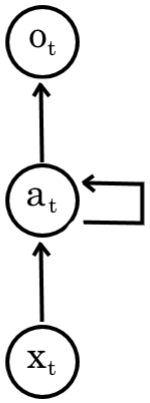

Figure 2.1: Simple recurrent neural network with one hidden unit

The network can be unfolded for each time step and can then be interpreted as a general neural network, as shown in Figure 2.2.

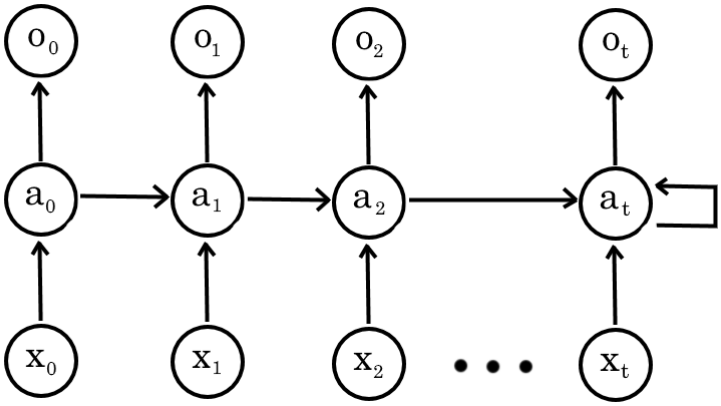

Figure 2.2: Simple recurrent neural network with one hidden unit unfolded over multiple time steps

A recurrent neural network with a single hidden layer can be defined with an input layer containing N_{input} number of neurons, a hidden layer containing N_{hidden} number of neurons and an output layer containing $N_{classes}$ number of neurons. The weight matrix $\mathbf{W}_{hidden,input}$ contains the weights for the connections between the input and the hidden layer and has dimensions $N_{hidden} \times N_{input}$. The matrix containing all weights of the edges connecting the hidden with the output layer is denoted by $\mathbf{W}_{output,hidden}$ and has dimensions $N_{classes} \times N_{hidden}$. A third weight matrix is needed for the feedback loops. Let $\mathbf{W}_{recurrent}$ be a weight matrix of dimensions $N_{hidden} \times 1$, and b_{input} and b_{hidden} be the bias units of the input and hidden layer, respectively. The recurrent neural network can then be formulated recursively for time steps $t = 1, \ldots, T$:

$$\begin{aligned}
\mathbf{a}_{[0]} &= \mathbf{a}_{init}, \\
\mathbf{a}_{[t]} &= act(\mathbf{W}_{hidden,input}\mathbf{x}_{[t]} + \mathbf{W}_{recurrent}\mathbf{a}_{[t-1]} + b_{input}), \\
\mathbf{o}_{[t]} &= act(\mathbf{W}_{output,hidden}\mathbf{a}_{[t]} + b_{hidden}),
\end{aligned}$$

where $mathbfx_{[t]}$ denotes the input at time step t, $\mathbf{a}_{[t]}$ denotes the vector containing all cell activations at time t, $\mathbf{o}_{[t]}$ denotes the vector containing all cell outputs at time t, and T is the length of the data sequence. These recursive properties allow the recurrent network to adapt to time dependent properties of the data, which makes it a great candidate for gesture recognition.

LSTM Cell In order to add long-term memorization abilities to a recurrent neural network, Hochreiter and Schmidhuber [HS97] developed the Long Short-Term Memory (LSTM) cell. A single LSTM cell represents a neural network layer in itself. The cell holds a cell state, which is capable of storing information for longer periods of time. This allows for the recognition of patterns that span across multiple time steps. The size of the hidden cell state is given by the number of specified hidden units within the cell. The resulting output of a LSTM cell is therefore given by the number of hidden units. The cell state is controlled via 3 gates:

- **Input Gate**: The input gate controls which parts of the new information is going to be stored in the cell state.
- **Forget Gate**: The forget gate controls which parts of the cell state is going to be thrown away.
- **Output Gate**: The output gate computes the output of the cell, which is then send to the next cell in the chain.

Each gate is connected to weights, which take part in the learning process and are thus also trained and updated. A neural network using LSTM cells is capable of developing long-term memory, which allows it to recognize patterns in sequential data even better.

2.2.2 Convolutional Neural Networks

All machine learning algorithms heavily depend on the applied feature extraction process [AH17]. Historically, these features have tediously been handcrafted using techniques like scale invariant feature transform, bag of features or bank of Gabor filters. With the advent of deep learning, convolutional neural networks changed the way these tasks are approached. These kind of networks are able to learn how to extract meaningful features without having to craft them by hand. Convolutional networks are heavily used in image recognition tasks, for example obstacle recognition in autonomous driving systems [Sam18].

Structure Convolutional neural networks fulfill two tasks: feature extraction and classification. The structure of these networks can also be split into these two tasks. For feature extraction, a convolutional neural network uses a number of convolutional- and pooling layers. For classification, the network uses fully connected layers, also often called dense layers. Dense layer work similar to output layers of a conventional feed-forward neural network.

An often used example for the purpose of convolutional neural network is the classification of hand written digits. The MNIST dataset [LCB] contains 28×28 pixel sized images of hand written digits. In order to extract features the convolutional layer of an neural network takes an image as an input and apply a series of filters to it. A filter can be defined by its dimensions $dim_x \times dim_y$, its step size $step_x \times step_y$ and its weight matrix of dimension $dim_x \times dim_y$.

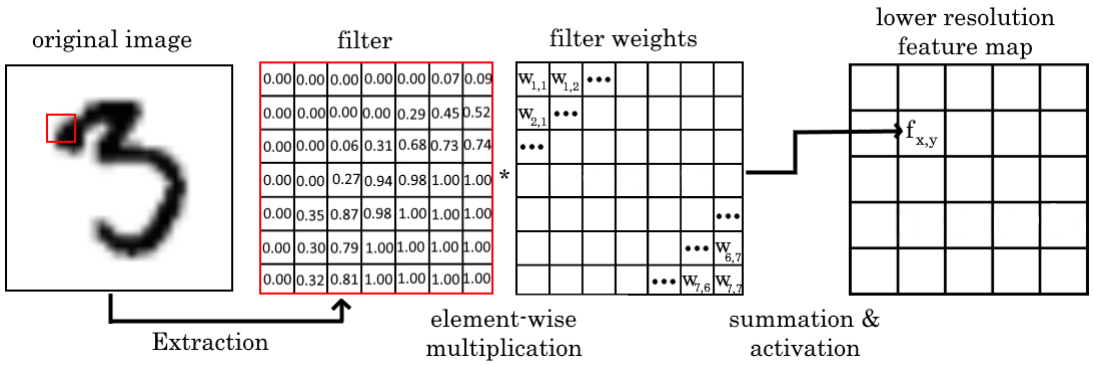

Figure 2.3: Convolutional layer filter step

The filter is applied on the input and acts like a window, which restricts the part of the image which is considered at each step. This process is displayed in Figure 2.3. The filter operation is an element-wise multiplication of the input values within the window with the corresponding weights of the filter. A single operation produces one scalar value. The window then moves along the input data with the corresponding step size. This method produces a feature map of lower dimensions. Multiple filters can be applied, each producing its own feature map, resulting in an amount of N_{filter} feature maps.

In order to further reduce the features, a pooling step can be applied. By minimizing the size of each feature map, the amount of operations in the next steps decreases drastically. Common pooling methods work much like a filter, an example being max pooling. In max pooling, a window is applied to the feature map. A pooling window of dimension 2×2 will look at four values each step. The highest value within the pooling window is chosen as the output value. This particular method will cut the size of the data in four. Depending on the structure, a variable amount of convolutional and pooling steps can be applied before reaching the fully connected (also called dense) layer. The input of the dense layer will be the values of each feature map extracted through the multiple steps of the convolutional and pooling layers. The fully connected layer acts exactly as a conventional feed forward neural network with weight matrices and output nodes.

2.3 Learning and Training

There are different factors that influence the way a neural network learns: activation function, error or loss function, regularization, and optimizer. The functionality and different choices for each component are explained and discussed in this Section.

2.3.1 Activation Functions

After each node has processed its input, the activation function is applied. The goal of this function is to indicate "how activated" the node is in its current state as an effect of the current input. The second responsibility of an activation function is to clamp a node's output value within a certain interval. The sigmoid function is a common choice as it outputs values between zero and one. Other popular activation functions for neural networks are the rectified linear unit function (ReLU) and the tangens hyperbolicus (tanh). The rectified linear unit function is defined as

$$reLU(z) = max(0, z),$$

where z is the value of the relevant node after processing the input with

$$z = \mathbf{W}\mathbf{x} + b.$$

ReLU is a frequently chosen activation function since it counteracts the vanishing gradient problem and promotes sparsity [FBB11].
The tangens hyperbolicus is defined as

$$tanh(z) = 2 \cdot \sigma(2z) - 1,$$

where σ is the sigmoid function defined as

$$\sigma(z) = \frac{e^z}{1 + e^z}.$$

tanh is preferred over the sigmoid function, as its gradient has a much larger range than the gradient of the sigmoid function, thus also counteracting the vanishing gradient problem.

2.3.2 Error or Loss Functions

The task of an error function is to determine the distance between the predicted output of a neural network and the expected output for the network given by the label. The crossentropy is usually chosen as the error function, as most neural networks are trained using the maximum likelihood approach commonly found in logistic regression. The output values lie between 0 and 1; hence, the values could be interpreted as probabilities. To ensure that the sum across all probabilities is 1, the softmax function is applied. The probability that input $\mathbf{x}$ belongs to class c is thus defined as

$$P_{x,c}(y = c|\mathbf{x}, \Theta) = softmax(c, \mathbf{out}),$$

where **out** denotes the output values of the nodes in the final layer. The softmax function is defined as

$$softmax(c, \mathbf{out}) = \frac{exp(\mathbf{out}_c)}{\sum_{k=1}^{K} exp(\mathbf{out}_k)}.$$

where k sums over all possible classes, and K is the total number of classes.

Let $b_{x,c}$ be a binary operator which returns 1 if c is the correct classification for x, and 0 else, formally

$$b_{x,c} = \begin{cases} 1, & \text{if } Class(\mathbf{x}) = c \\ 0, & \text{otherwise} \end{cases}.$$

A separate loss is calculated for each class and then added together:

$$-\sum_{c=1}^{C} b_{x,c}\, log(P_{x,c})$$

To get the output of the error function, the loss of each individual example needs to be summed up and averaged:

$$E_{CE}(\Theta) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} b_{x_n,c}\, log(P_{x_n,c})$$

2.3.3 Regularization

Regularization is a term used to describe the act of preventing weights from reaching too high values in order to prevent overfitting. Overfitting occurs when a model adapts too strongly to the data, thus classifying new inputs wrongly. A model is especially prone to overfitting if the data contains outliers.

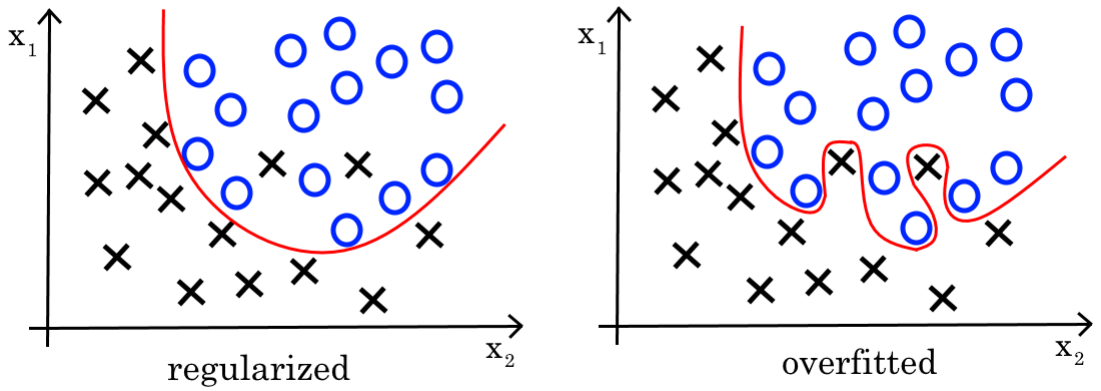

Figure 2.4: A regularized vs. an overfitted decision boundary

The decision boundary in figure 2.4 shows clear signs of overfitting, as the boundary is strongly warped in order to include each outlier. New datapoints could be falsely classified

as a cause of the warped decision boundary. The model thus loses its ability to generalize well. Preventing the weights of reaching too high values prevents the decision boundary from becoming too warped. The model is then less likely too overfit. This preventive method is called regularization and is commonly implemented at two different points in the model: dropouts and the loss function.

Loss Function Regularization As mentioned, a common choice for neural networks is the cross entropy error function. In order to add regularization to the loss function, another summand is added to the function. The role of this summand is to punish high values of weights. A common choice for this is the L2-regularization defined as

$$L2 = \lambda \sum_{i=1}^{N_{weights}} w_i^2,$$

where λ is the regularization rate, which can be tweaked to adjust the regularization's strength. This regularization term is then added to the error function to enforce regularization.

Dropouts Dropouts are another way of counteracting overfitting [Bud16]. Dropouts refer to the technique of leaving out certain nodes randomly when training a neural network. A probability p is defined for this purpose. When training a neural network, at training step, each node has a chance of $(1 - p)$ to be kept for training. The resulting network is a reduced version of the old one. Neural network nodes tend to develop a co-dependence, which results in less robust classifiers. By dropping out certain nodes of a network, the network needs to learn more robust features, which results in a more robust classifier.

2.3.4 Optimizer

The goal of training a model is to minimize the error function by adjusting the weights. The goal model Θ^* can be defined as

$$\Theta^* = argmin_\Theta(E_{CE}(\Theta)).$$

Minimizing the error function is not trivial since the complete function of a neural network is often complex and not convex, making the use of common analytical approaches difficult. Algorithms, which solve this minimizing problem are called optimizer. In order to find the best model, optimizers need to repeat two basic iterative steps:

- Backpropagation: Finding the partial derivatives of the error function
- Learning step: Adjusting the weights based on the found derivative

The algorithms tested in this thesis are Adam optimization [KB17], Adadelta optimization [Zei12], Adagrad optimization [DHS11], and RMSProp optimization [Hin].

2.4 Tools

2.4.1 Tensorflow

The neural network implementation is done using Tensorflow 1.8.0. Tensorflow is an open source machine learning API developed by Google [Goo]. Tensorflow allows high-performance computing across multiple platforms including CPUs and GPUs. Once an abstract graph of the model has been defined within tensorflow, the API seamlessly scales the computation up according to the available processing power. The tensorflow estimator API will be used as it provides a good mix of high and low-level control. The model can still be implemented on a low level, while the training process gets sped up with great high-level control. Summaries and checkpoints can be created while training for an easy way to visualize the training process via the built in visualization tool.

2.4.2 Unity Game Engine

Unity is a game engine developed by Unity Technologies. Unity is a licensed game engine, which features a free version for non profit projects. Unity's script language is C# or Javascript. For this thesis, C# was used. Unity engine outdoes other engines in terms of its accessibility and its adaptability to modern interactive technology such as common AR and VR applications or the Myo armband.

3 Gesture Recognition Task

3.1 Overview

The recognition task will consist of classifying three different hand gestures using two different neural network architectures. The three hand gestures in questions are the rock, paper, scissors gestures from the commonly known rock-paper-scissor-game. Moreover, a fourth gesture called the idle or relaxed gesture is added to the classification task. This prevents the classifier from making arbitrary choices if no gesture is performed. The reasoning behind picking exactly these gestures is threefold. Firstly, the gestures are simple but still easily distinguishable. Secondly, the gestures are well known and intuitive to anyone. This proves especially useful in the data recording process. Thirdly, the parameters used for the convolutional neural network architecture discussed in this thesis are based on the model proposed by Goll [Gol18]. He develops a convolutional neural network architecture for hand gesture recognition, using the rock-paper-scissors gestures as the classification task as well. As both networks are specialized to recognize exactly the same gestures, a meaningful comparison can be made. Data has been recorded from 15 different volunteers. Each participant recorded 100 rock, 100 paper, 100 scissor and 100 idle gestures. Noise is expected to have been introduced due to human error during the recording sessions. The data has then been used to train both neural network models. Both networks are evaluated on different metrics among accuracy and ease of training. Finally, the architectures are compared with each other.

3.2 Recording Environment

Unity engine was used to create a gesture data recording application in which each volunteer recorded 400 different gestures. Within Unity, a scene was created which functions as the main hub for starting the recording process. Each user identifies with a username which is used for naming the dataset, allowing the user to record the gestures across multiple different sessions. In the next menu, the user is able to choose from three different training options: “Manual Training”, “Rhythmic Training Slow” and “Rhythmic Training Fast”.

Manual Training Within the manual training mode the user is able to input gestures arbitrarily by performing the gesture while inputting one of the three specified buttons on the keyboard. Each button represents a gesture. The application is set up to manage a circular buffer, which contains the Myo data of the last 40 frames. This buffer is continuously updated each frame, pushing the new data of the current frame into the buffer and dropping the data of the oldest frame. The reason for choosing a buffer size of exactly 40 frames is elaborated in Section 3.3.2. On a recognized button input, the

program will safe the buffer after a certain delay. Again the chosen length of the delay will be discussed later as well. The saved data is written into a .csv file, along with a label for the recorded gesture and a unique ID. This training is intended for debugging purposes.

Rhythmic Training The idea of the rhythmic training environment comes with the necessity of precisely labeling the training data. In the case of gesture recognition, a sequence of data points are examined instead of single data points. Recording and labeling data thus becomes less trivial. The rhythmic training environment provides a fun, intuitive way of recording data while also skipping a lot of manual labor and keeping the proportions between the number of different gestures equal.

The inspiration for the rhythmic training environment derives from popular rhythm games like "Guitar Hero" [1] or "Osu!" [2]. The player's task is to hit specified buttons to the rhythm of background music. The player is aided in doing so by visual clues, which are in sync with the beat.

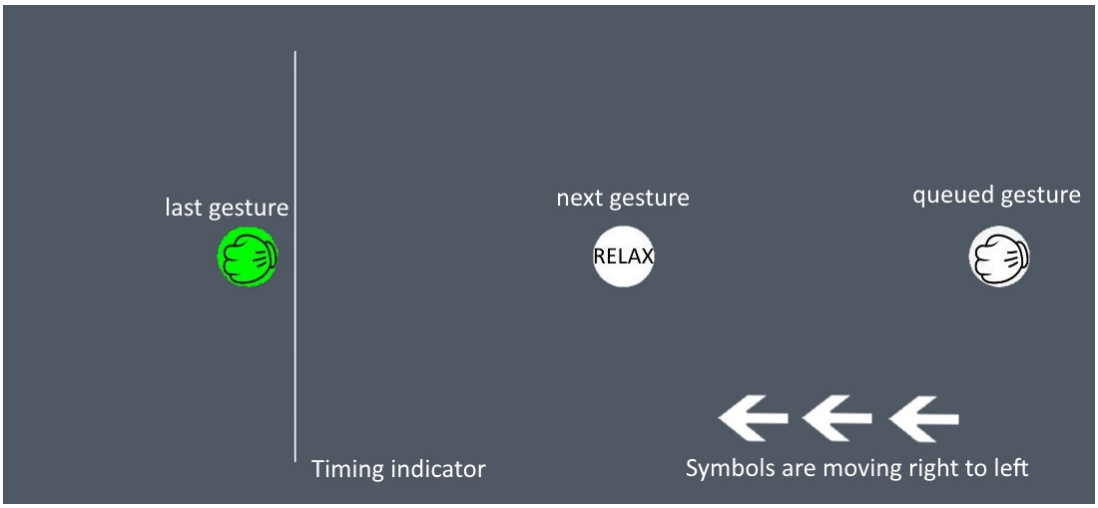

Figure 3.1: Screenshot of the rhythmic training with additional annotations

When starting the session, a 90 beats-per-minute fast drum beat will start playing. Symbols will appear from the right of the screen, as shown in Figure 3.1. There are 4 different kinds of symbols [3], each representing one of the 4 available gestures: rock, paper, scissors, idle (Figure 3.2).

[1] https://www.guitarhero.com/de/
[2] https://osu.ppy.sh/home
[3] https://www.iconspng.com/image/36942/rockpaperscissors

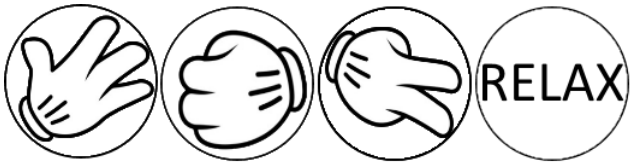

Figure 3.2: Symbols of all 4 gestures

The players task is to perform the gestures specified on the symbols in the order as the symbols appear. The timing is set with a visual cue. On the left part of the screen is a white vertical line. In the moment the symbol crosses the white line, the player needs to perform the specified gesture. This moment is further highlighted by being in sync with the background beat. This small game has the following positive effects:

- It is more engaging for the player, as the player needs to focus on the symbols and the music in order to correctly perform all gestures.
- The game allows for full control over how many symbols of each gesture appear, allowing the automatic balancing of the gesture distributions.
- The timing of each symbol allows for a fully automated way of processing the data. The timing can be used to automatically save the relevant time slice of the continuous data flow. On top of that, the gesture specified by the symbol can be used to label the processed data slice.

Furthermore, the frequency in which the symbols appear can be altered by choosing the slow or fast version of the rhythmic training. The difference is that symbols appear every 2^{nd} or 4^{th} beat in the fast or slow version. This helps keeping the recording session engaging for fast players, while also providing an easier version for slower players. The recording session should be finished without making any errors, so the difficulty of the game should remain easy overall. Each session contains 40 gestures, 10 of each type. Each participant completed 10 sessions, resulting in 400 gestures for each participant.

3.3 Data

3.3.1 Structure

As explained earlier, only the surface electromyographic(sEMG) sensors are chosen for usage, as one of the goals is to estimate the usefulness for sEMG data for classifying hand gestures. It is assumed, that the motions sensors would have hardly contributed to the classification accuracy, as each rock paper or scissors gesture shares the same general motion when looking at the lower arm. The output of the eight sEMG data channels is saved together with the corresponding gesture label as well as with a a unique ID for each action within a session. The resulting table is stored as a .csv file as shown in Table 3.1. The action ID was only used for debug purposes and is willingly omitted for the training of the neural networks.

EMG1	EMG2	EMG3	EMG4	EMG5	EMG6	EMG7	EMG8	Gesture	ActionID
-4	-28	127	-1	2	1	1	-3	1	0
1	2	6	-5	-5	-3	-1	2	1	0
-15	45	-94	3	3	0	2	5	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.1: Example snippet of a data sequence

3.3.2 Recording Parameters

Two important parameters had to be chosen for the recording. The first one is the length of one data sequence containing one gesture. The other is the offset of the sequence window in relation to the timing set during the recording session. When recording 40 frames of data, for example, it is important to specify which 40 frames in relation to the timing are saved. The 40 frames following the timing might not be the most relevant ones.

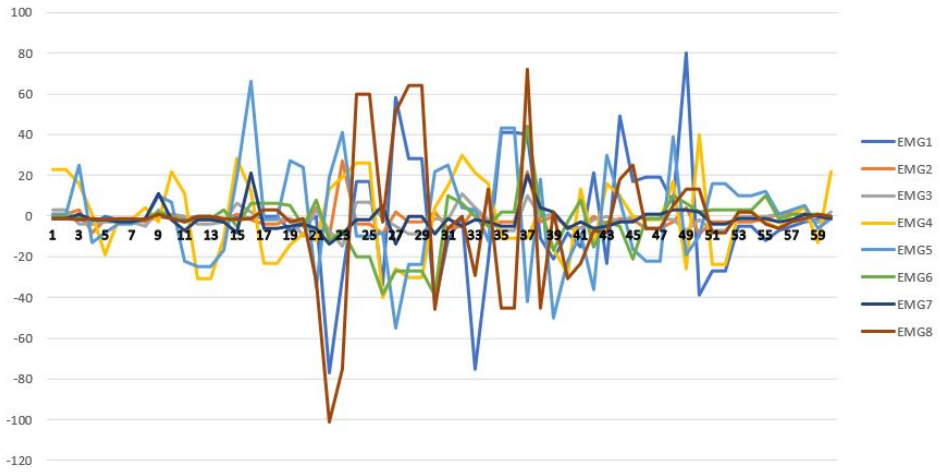

Figure 3.3: Example rock gesture data (60 frames)

Figure 3.3 shows an example recording of a rock gesture across 60 frames. The relevant part of the data seems to occur within a 40 frame window between the 15^{th} and the 55^{th} frame. The timing given by the rhythmic training environment occurred at the 30^{th} time step. The same behavior was observed for multiple examples. Therefore, the sequence length was set to 40 frames. In order to capture the relevant data, the 15 frames before the timing and 25 frames after the timing are recorded.

3.3.3 Datasets

Since the data recording was based on timing, the data was prone to have some errors as some volunteers accidentally inputted the wrong gesture or missed the timing, thus making the recorded gesture data less usable for training. In order to counteract the human error,

the datasets produced by each volunteer individually was ranked by quality. For this, a recurrent and a convolutional neural network were trained a single dataset produced by one volunteer. Then the accuracy was measured on an evaluation set, created from the data produced by the same volunteer, whose data was used for the training set. This process was repeated for each dataset. The accuracies resulting from these test were then used to rank the datasets.

The idea behind this procedure was that good, consistent data from a single source will produce a robust network that is able to classify new data from the same source. Finally, the best five datasets were put together to make up the "Top5" dataset. Same with the best 10 datasets resulting in the "Top10" dataset and likewise the best 15(all data) for the "Top15" dataset. The simple recurrent neural network used a recurrent layer consisting of 64 bi-directional LSTM cells ending in a four class logits output layer. The convolutional neural network architecture used corresponds to the architecture proposed by Goll [Gol18].

3.4 Metrics

The metrics which are used to evaluate and compare the final models are as follows:

- Highest accuracy
- How many steps until 90% accuracy
- Complexity
- Computational and prediction latency
- Generalization
- Specialization

The metrics are now further explained and justified.

Highest Accuracy Accuracy might be the simplest and most important metric. It measures how well the model is classifying incoming data. This metric is evaluated by taking the number of correctly classified data examples over the total number of predicted examples:

$$accuracy = \frac{N_{correct}}{N_{total}}$$

The data used to evaluate the accuracy is taken from the evaluation set. The evaluation is a part of the overall data, but completely disjoint from the training set. It is important that the evaluation set contains an equal number of examples across all gestures as well as a mixed amount of data from different persons.

How many steps until x% accuracy For this metric, earlier trained models are evaluated on the property of "how many training steps were needed to reach x% accuracy". This metric hopefully gives some intuition about how fast a model is learning. This is interesting to find out, as training a neural network can prove time consuming and needs data.

Complexity The complexity of a neural network is given by its structure. Convolutional or pooling layers add additional complexity, recurrent layers introduce additional forward edges and dense layers with many weights are needed for classification. The complexity of a neural network is difficult to discuss. Examples of factors that have an effect on the complexity include the number of layers, number of hidden units within the layers, and the number of classes.

The complexity has three impacts. First, the more operations a neural network needs to perform, the longer the training process will take.

Second, energy consumption is an important measurement, as mobile devices are limited by their battery life. More operations per prediction means that the device's power supply will run out sooner. In a mobile environment, low complexity models might prove superior.

Lastly, the complexity has an effect on the latency of the model's predictions. Since a prediction can only be made after the model has finished all of its calculation, the number of total operations becomes important. A model with fewer operations will be able to make predictions faster.

As complexity is hard to pin down as a number, the complexity of different models will be discussed by estimating the number of weights for each network. Furthermore, the computational latency also provides a good indication of the complexity of a model.

Computational and Prediction Latency The computational latency is the time difference between the arrival of the input and the network's output, formally

$$latency_{computational} = t_{output} - t_{input}.$$

To get a precise result, the time needed for each model to predict 1000 training examples is measured five times, then averaged and finally divided by 1000. The processing unit's power affects this metric, which is why all tests have been made using the same hardware.

Another kind of latency occurs when doing predictions on a sequential data. The prediction latency is the difference between the time a gesture is performed and the time the gesture is recognized, formally

$$latency_{prediction} = t_{correct\ prediction} - t_{gesture\ started}.$$

For real time applications, the responsiveness of the input devices is key for a natural and convenient interaction between the user and the machine. High prediction latency means bad responsiveness and the application may consequently become frustrating to use or even completely unusable.

Generalization In machine learning, generalization means how well the model can classify examples outside the training and evaluation set, preferably even from different sources than the ones used to record the training and evaluation sets. This is important as the training of the network happens inside the confined world represented by the training and evaluating set. It is possible that a model with high accuracy on the evaluation set still generalizes poorly. This is usually a good indicator that the dataset is a bad representation of the real world.

First the recurrent model is trained using data created from 5 different persons. Then the best performing model is chosen. The model will be evaluated using data taken from persons outside the training set. The accuracy is then used as the generalization metric.

Specialization In some cases it is not required for a model to generalize well, but instead to focus solely on specialization. An example of such a scenario is a health assistant device which measures the heart rate of the user. As the device would not be passed on to others and as it is worn around the clock, a huge amount of data can be acquired for training purposes. The resulting model would be a very specialized model that is best suited to classify new data from the same source but generalizes poorly.

To measure the specializing capability of a model, the model will be trained using a large amount of data from only one user source. The accuracy over the evaluation set, taken from the same source then gives a good indication about how well the model can specialize.

4 Recurrent Neural Network Architecture

One of the goals of this thesis is to build a robust recurrent neural network architecture for gesture recognition. To find a suitable model, many parameters have been considered and tested. The implementation and testing was done using tensorflow.

4.1 Parameters

Recurrent Cell Type For a recurrent neural network model, the parameter decisions start at the single nodes. Tensorflow provides 5 different types of recurrent neural network nodes:

- Simple recurrent cell
- LSTM recurrent cell
- Bi-directional LSTM recurrent cell
- GRU cell
- Multilayered recurrent cell

The usefulness of each cell type was assessed during testing and will be discussed in the next Section 4.2.

Total Number of Recurrent Cells The second important parameter is the total number of recurrent cells used. A smaller number of cells allows for a faster training and learning, with a cost of lower accuracy. A high number of cells causes long training phases and will not necessarily result in higher accuracy. Further, the model's complexity will be increased, resulting in higher latency and energy consumption. The optimal number of recurrent units was assessed together with the optimal cell type during testing. To avoid confusion, in case of LSTM and GRU cells, the total number of cells refers to the total number of hidden units within the LSTM- or GRU cell.

Activation Function Activation functions change the way a neural network node responds to its input. Not only classifications change depending on the activation function, but the learning process can change as well as the learning step is dependent on the activation function's derivative. Three different activation functions are tested in this Chapter: ReLU, tanh, and the sigmoid function.

Loss Function and Regularization As explained in Section 2.3, loss function and regularization are important to build a robust model that will not overfit. The cross entropy function will be used as a loss function and the usefulness of additional regularization will be tested.

Optimizer Optimizers are algorithms used to find the best weights in order to minimize the overall loss. Three different optimizers are evalualted in this Chapter: Adam Optimizer, AdaDelta Optimizer, and RMSProp Optimizer.

Learning Rate The learning rate is a parameter passed to the optimizer, which controls the stepsize with which an optimizer updates the weights. A bad learning rate can cause the optimizer to never terminate, while a appropriate learning rate can increase the overall achieved accuracy.

4.2 Structure Tests and Parameter Choices

4.2.1 Cell Type and Total Number of Cells/Units

The first round of testing was used to get an overview on the effects and usefulness of the five different available cell types and the chosen number of units in the recurrent layer. For this, a training and an evaluation set was created from the Top5, Top10 and Top15 dataset each. Multiple recurrent neural networks were trained, one for each combination of cell type and either 16, 64 or 128 units in the recurrent layer. Each network has been trained three times, one time for each data set. 100 repetitions of the training set were used for training. The results for each dataset are summarized in the Tables 4.1, Table 4.2, and Table 4.3.

# units	Simple Rec.	LSTM	Bi-dir. LSTM	GRU	Multi-LSTM
16	0.4500	0.8120	0.8110	0.8353	0.6829
64	0.4200	0.8800	0.8986	0.8764	0.9211
128	0.4700	0.9000	0.9057	0.8829	0.9094

Table 4.1: Accuracy using various cell types and number of hidden units on the Top5 dataset

# units	Simple Rec.	LSTM	Bi-dir. LSTM	GRU	Multi-LSTM
16	0.6438	0.7685	0.7900	0.7877	0.8169
64	0.5069	0.8338	0.8325	0.8162	0.8785
128	0.4800	0.8415	0.8462	0.8338	0.8823

Table 4.2: Accuracy using various cell types and number of hidden units on the Top10 dataset

# units	Simple Rec.	LSTM	Bi-dir. LSTM	GRU	Multi-LSTM
16	0.5582	0.7321	0.6973	0.7217	0.8288
64	0.4625	0.8237	0.7511	0.7152	0.8158
128	0.4826	0.8332	0.7652	0.7429	0.8348

Table 4.3: Accuracy using various cell types and number of hidden units on the Top15 dataset

Three important pieces of information were gathered from the first test round:

- The simple recurrent cell generally performs worse in most cases compared to the other cell types.
- The GRU cell's highest accuracy on each dataset is lower than the accuracy of other cell types besides the simple recurrent cell.
- Increasing the total number of units up to 128 improves the accuracy of any cell type except the simple recurrent cell.

The second test round only considered the three cell types: LSTM cell, bi-directional LSTM cell and the multilayered LSTM cell. Only the Top5 dataset was used for testing as training proved faster and the results were more accurate. A new set of evaluation and training set was created from the dataset. New models were created from all combinations of the three chosen cell types and either 16, 64, 128, 256 or 512 units in the recurrent layer. Each model was trained three times on the same training set with 100 repetitions. The multilayered LSTM cell model was only trained one time, as training proved very time consuming. The results of the tests are summarized in Table 4.4.

# units	LSTM 01	02	03	Bi-dir. LSTM 01	02	03	Multi 01
16	0.8471	0.8343	0.8329	0.8257	0.7967	0.7667	0.8671
64	0.8886	0.8857	0.9014	0.8810	0.8947	0.8738	0.9100
128	0.9186	0.9029	0.8900	0.8861	0.9025	0.9171	0.9114
256	0.8857	0.8971	0.9071	0.9043	0.8914	0.9100	0.9100
512	0.9214	0.9114	0.9243	0.9229	0.9114	0.8986	0.9086

Table 4.4: Accuracy of the second test run on the Top5 dataset

In the second run, the best results were achieved using the LSTM cell type with a total number of 512 units. The model achieved 92.42% accuracy in the third run and 92.14% in the first run. The second best scores were the result of the bi-directional LSTM cell type model, again with 512 units in the recurrent layer. Accuracy of 92.29%, 91.14% and 89.86% was achieved. It is noticeable however, that the model seems less robust in the early training phase as seen in Figure 4.1.

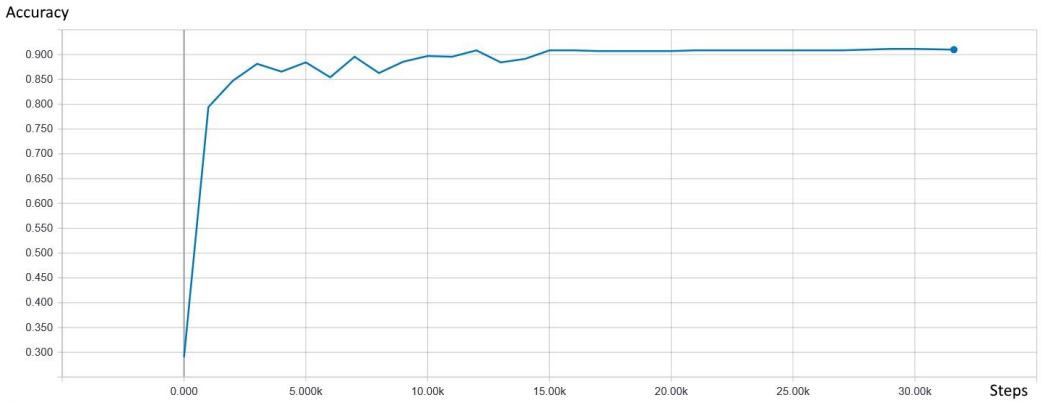

Figure 4.1: Accuracy of the 512 unit bidirectional LSTM model

The multilayered LSTM cell model achieved its best accuracy of 91.14% when using 4 layers with 128 cells each. As the best accuracy for the LSTM model was achieved using 512 units and the best accuracy for the multilayered model was achieved using 4 layers with 128 units each, further testing will be done to distinguish. The bi-directional cell type will be dropped from consideration. Even though the cell type adds additional complexity to the network, no gain in accuracy is observed compared to the LSTM model. Further, the bi-directional cell type scored lower on the Top10 and Top15 dataset in the first run, indicating that it performs worse on a more variable dataset. The next two tests are done to investigate the following hypotheses:

- Increasing the number of hidden units will further increase accuracy.
- A multi-layered model with the same total number of units across all layers will perform better than an LSTM cell model.

The first hypothesis is made because in every test, the models with the highest number of hidden units performed the best, and hence further increasing the number might improve the model. For this reason, a LSTM cell type model is trained with 1024 hidden units. The training and evaluation data used is the same as in the second training round. The training process took considerably longer than any other model so far. The training progress indicates that the model is less robust. It achieves a peak accuracy of 93.00%, but the training progress is filled with rough spikes as seen in Figure 4.2. Considering the added complexity and the longer training duration for almost no considerable accuracy gain, the model will not be used.

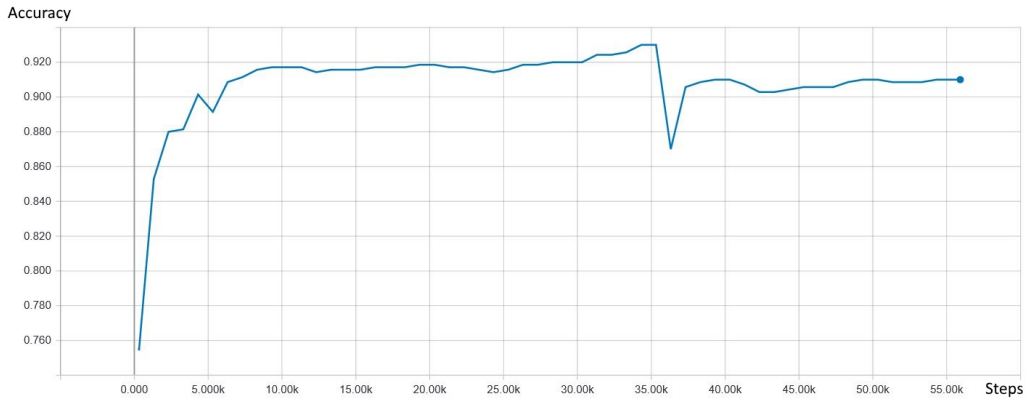

Figure 4.2: Accuracy of the 1024 unit LSTM model

To test the second hypothesis, a multi-layered recurrent neural network with 2 layers using 256 units each has been trained on the same dataset. After training the network on 100 repetitions of the training set, the best resulting model reached 93.57% accuracy while remaining very robust as shown in Figure 4.3.

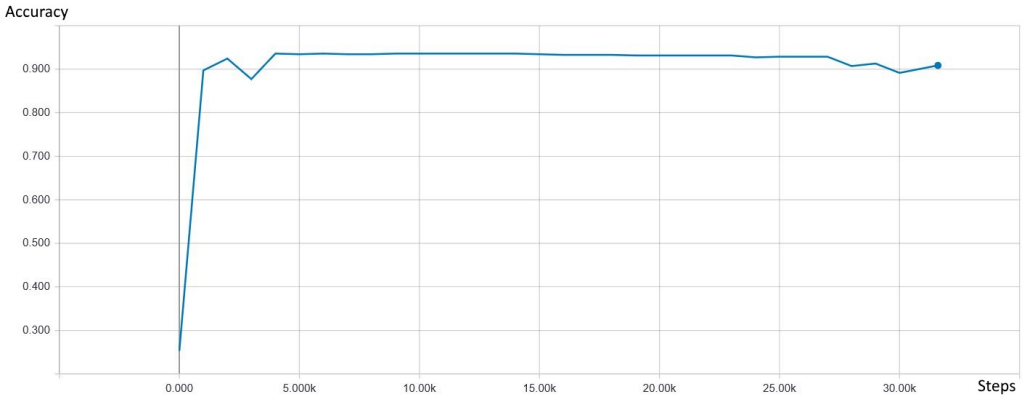

Figure 4.3: Accuracy of the 2-layered, 256 units each, LSTM model

For the final architecture a version of the multi-layered recurrent neural network model will hence be considered. A few more test runs were done, to figure out the best combination of number of hidden units and layers. The test results are summarized in Table 4.5.

# layers	# units	01	02	03
2	256	0.9357	0.9243	0.9186
2	200	0.9043	0.9171	0.9143
2	300	0.9229	0.9129	0.9229
3	170	0.9071	0.9057	0.9157
4	128	0.9114	0.9157	0.9143

Table 4.5: Accuracies of various models with around 500 total units each

Five different combinations are tested, each containing a total number of about 500 hidden units. The combinations tested are three variations of a 2-layered model with 200, 256 and 300 units in each layer respectively, one 4-layered model with 128 hidden units in each layer, and lastly a 3-layered model with 170 units in each layer. The best scoring model was still the 2-layered model with 256 units, however, using 300 units in a 2-layered model also produced good results. Due to the added complexity, the 300 unit model is considered worse.

4.2.2 Activation Function

With a newly generated testing and evaluation set, the 2-layered 256 unit architecture was tested using three different activation functions: tanh, ReLU, and the sigmoid function. The default activation function, which the model was using in earlier tests, is the tanh activation function. The next tests were done to examine whether changing the activation function yields better results or not.

ReLU The first three test runs were done on the proposed architecture using the ReLU activation function. In all three runs, the model seemed to adapt a good accuracy at about 80% but then the accuracy dropped and never recovered as seen in Figure 4.4.

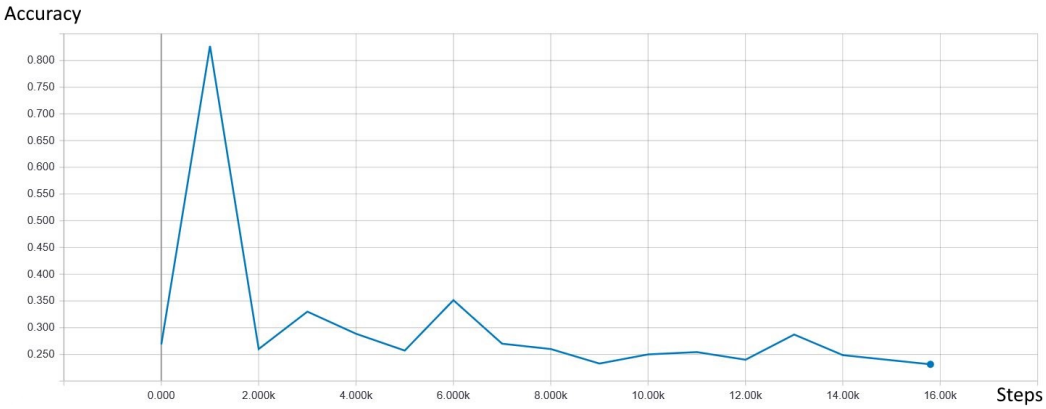

Figure 4.4: Accuracy of the model using the ReLU activation function

This occurrence was observable on all three test runs. This might be due to the fact that the ReLU activation function can return values greater than 1. The gates of a LSTM cell can be interpreted as filters on "how much" of the input signal is being used to update the cell, where 1.0 corresponds to 100%. By receiving values greater than 1, the incoming signal can be amplified, which in turn can make the model diverge. Thus, the ReLU activation function will not be considered for the final architecture.

Sigmoid and tanh Like tanh, the sigmoid function is another viable candidate for the role of the activation function in LSTM recurrent neural networks as the output ranges from 0 to 1. Two models, one using tanh and the other using the sigmoid function as its activation function were trained three times. The results are displayed in Table 4.6.

	Test01	Test02	Test03
tanh	0.9386	0.9371	0.9400
sigmoid	0.9143	0.9257	0.9314

Table 4.6: Accuracies of the model using tanh and sigmoid as activation function

Compared to the results of the tanh activation function, however, the sigmoid model falls short with an accuracy of 93.14% compared to the tanh model's top accuracy of 94.00%. For the final model, the tanh activation function is chosen.

4.2.3 Loss Function and Regularization

Even though the model did not show signs of overfitting, the usefulness of regularization was tested. Dropouts were not considered for testing as their usefulness for recurrent neural networks are usually limited to time series forecasting. L2-regularization, as described in Chapter 2.3.3, was implemented and tested in another three test runs. The results can be seen in Table 4.7.

Test01	Test02	Test03
0.9243	0.9186	0.9286

Table 4.7: Accuracy of the model using L2-regularization

The resulting models failed to reach an accuracy comparable to the tanh model from the previous section. The model seems to learn slower and the learning curve looks less robust (Figure 4.5) compared to the learning curve of the model without any regularization. The data is probably well distributed, which makes regularization unnecessary.

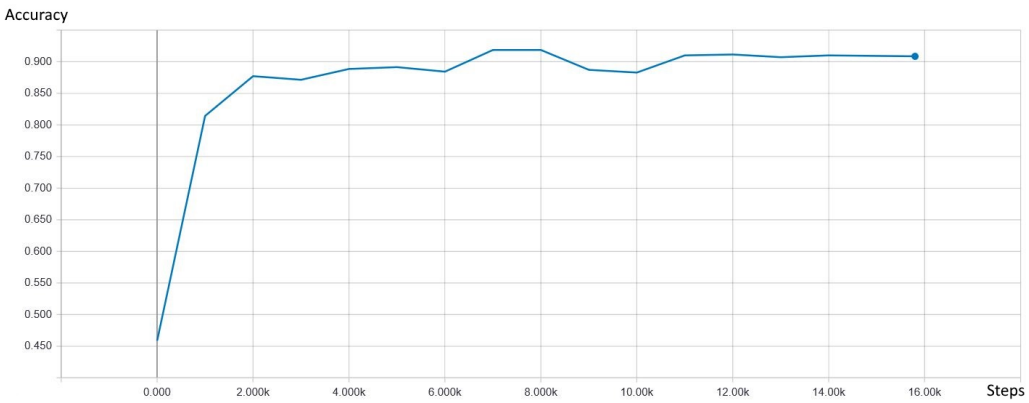

Figure 4.5: Accuracy of the model using L2 regularization

4.2.4 Optimizer and Learning Rate

For any test so far, tensorflow's implementation of the Adam algorithm was used in the form of tensorflow's AdamOptimizer. In this Subsection, two other optimizers, AdaDelta and RMSprop, are tested. The training data used for this test is the same data as in the activation function Subsection such that the results are comparable to the previous test results. The learning rate for all optimizer was set to $\alpha = 0.001$. Later in this Section, different learning rates are tested with the chosen optimizer. The results of the optimizer tests are summarized in Table 4.8.

Optimizer	Test01	Test02	Test03
Adam	0.9229	0.9157	0.9300
AdaDelta	0.7986	0.8143	0.8243
RMSProp	0.9186	0.9214	0.9200

Table 4.8: Accuracy of the model trained using different optimizer

After 100 repetitions of the datasets, the AdamOptimizer yielded the best accuracy. RMSprop converged fast, but the overall accuracy was still lower than AdamOptimizer with only 92%. The AdaDeltaOptimizer reached around 80% accuracy but after 100 repetitions the accuracy was still not converging. The learning rate was set to $\alpha = 0.001$. It was decided to do further testing using AdaDeltaOptimizer with higher learning rates.

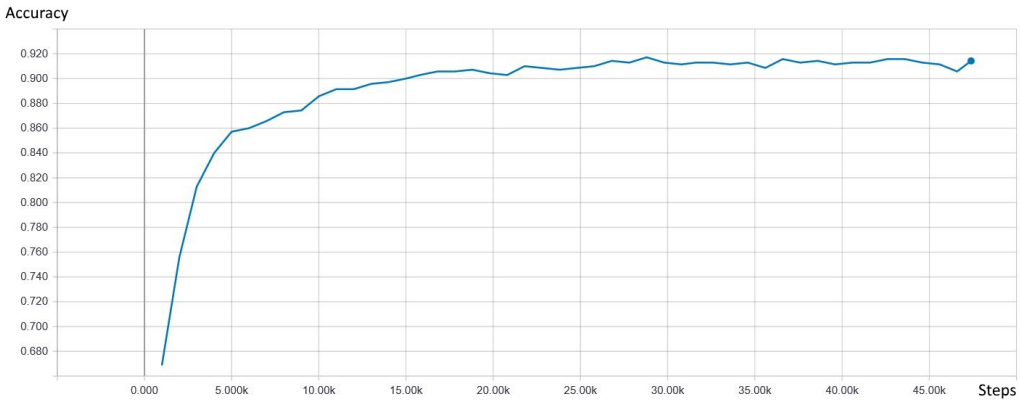

Figure 4.6: Accuracy of the model using AdadeltaOptimizer and $\alpha = 0.01$

Even after raising the learning rate to $\alpha = 0.01$ and letting the model train for 150 repetition on the dataset, the model's highest accuracy still remains at 91.71%, as seen in Figure 4.6. For the final architecture, the AdamOptimizer with a learning rate of $\alpha = 0.001$ will hence be chosen.

4.3 Final Structure

The final structure of the recurrent model uses LSTM cells. The input layer contains 40×8 nodes, as the input sequence is 40 frames long and the signal consists of 8 channels. Two LSTM cells containing 256 hidden units each are stacked on top of each other for the hidden part of the neural network model. The final layer consists of a fully connected logit layer containing 4 output nodes. The LSTM cells and their hidden units use tanh as their activation function. Loss is calculated without any regularization using the cross entropy loss function. Dropouts are not used for regularization. The Optimizer used for training is the AdamOptimizer using a learning rate of $\alpha = 0.001$. The final architecture is displayed in Figure 4.7.

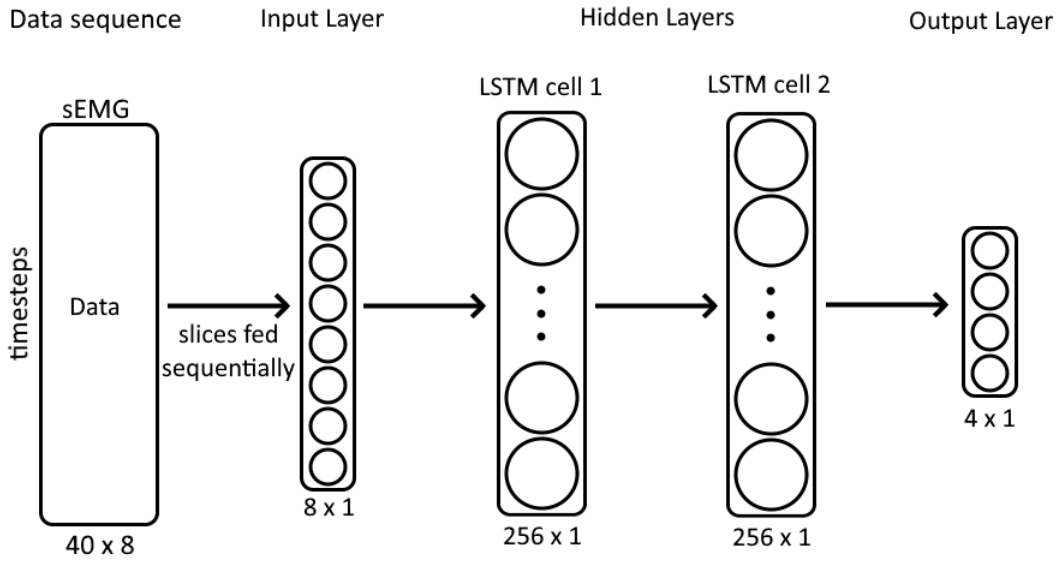

Figure 4.7: Structure of the recurrent neural network

4.4 Evaluation

The Evaluation is done using the methods described in Section 3.4. For fairness, each model will be trained on the same datasets within the different experiments.

Highest Accuracy To measure the highest accuracy, a set of training and evaluation data is created from the Top5 dataset. The final model is trained three times on the training set and then evaluated on the evaluation set. The results are shown in Table 4.9.

Test01	Test02	Test03	Highest Accuracy
0.9286	0.9157	0.9371	0.9371

Table 4.9: Evaluation of the highest accuracy

The model reached its peak accuracy at 93.71%.

How Many Steps until x% Accuracy To evaluate this metric, the training progress of the three models from the highest accuracy test are inspected. The goal was to find out how many training steps were required to reach 90% accuracy for the first time. The training steps needed for each model can be seen in Table 4.10.

Test01	Test02	Test03	Average steps required
25000	20000	20000	21666

Table 4.10: Training steps required to reach 90% accuracy

To reach 90% accuracy, the model took on average 4333 training steps.

Complexity The final structure of the recurrent neural network models consists of 3 parts: The input layer, the hidden layer and the output layer. The hidden layer can be further broken down into two LSTM cells, containing 256 hidden units each, stacked on top of each other. To get an overview over the complexity of the model, the total number of weights will be estimated. The number of weights used for an LSTM cell is dependent on the size of the input each time step N_{input} and the number of hidden units N_{units} within the cell. Neglecting the biases, the weights used for any LSTM cell can be estimated as follows.

$$N_{weights} = 4 \cdot N_{units}(N_{input} + N_{units})$$

The first cell in the final structure therefore uses a total number of

$$4 \cdot 256(8 + 256) = 270.366 \text{ weights.}$$

The second cell receives the output of the first cell as an input. The total number of weights used for the second cell is therefore

$$4 \cdot 256(256 + 256) = 524.288 \text{ weights.}$$

The output layer contains 4 nodes, each fully connected to the previous layer for a total of

$$256 * 4 = 1024 \text{ weights.}$$

Neglecting the weights related to bias nodes, the network uses an estimated number of about 800.000 weights. Important to note: to predict a given input, the network has to calculate each connection 40 times as the sequence length of our input is 40 frames long.

Computational and Prediction Latency The time it takes for the model to predict 1000 datapoints is measured in five different runs. The results are then averaged. The test results can be found in Table 4.11.

Test01	Test02	Test03	Test04	Test05	Average time required
16.3137s	16.4129s	16.1886	15.8448	16.3917	16.2303

Table 4.11: Computing time of the neurwl network for 1000 predictions

This means, that predicting a single data example takes about 0.01623 seconds or 16,23 milliseconds. With this rate, the network is able to predict just above 60 data examples per second, making it usable for real time applications.

The prediction latency was tested in a real time test. Continuous input was classified by the recurrent neural network. A gesture was then inputted simultaneous together with a button press. From the time of the button press, the time steps were counted until the classifier outputted the correct classification. Across all gestures, it took the model between 1 and 3 time steps to output the right classification, which is between 16.23 and 48.69 milliseconds considering the computational latency.

Generalization First, the recurrent neural network model is trained and evaluated on a the Top5 dataset three times. The highest accuracy model is chosen. The model achieved an accuracy of 92.71% as seen in Figure 4.8.

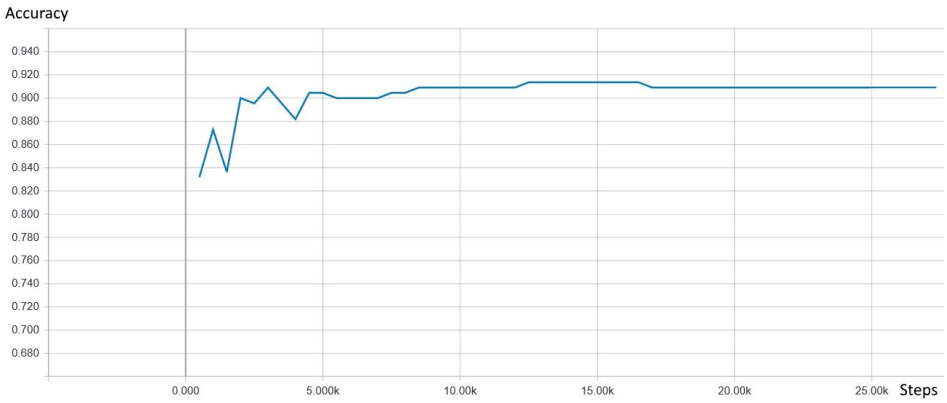

Figure 4.8: Accuracy of the chosen model

To assess the generalization capability, an evaluation set is created from the test data of 3 persons outside the Top5 dataset. The model reached an accuracy of 67.91% on the evaluation set.

Specialization For this metric another model has been trained. Data from only one person was used to train the model three times. The evaluation set stems from the same person but does not overlap with the training data. The best performing model was able to reach an accuracy of 97.86% as seen in Figure 4.9. The results of all test runs can be observed in Table 4.12.

Model 01	Model 02	Model 03	Highest Accuracy
0.9786	0.9643	0.9786	0.9786

Table 4.12: Accuracy on the specialized training data

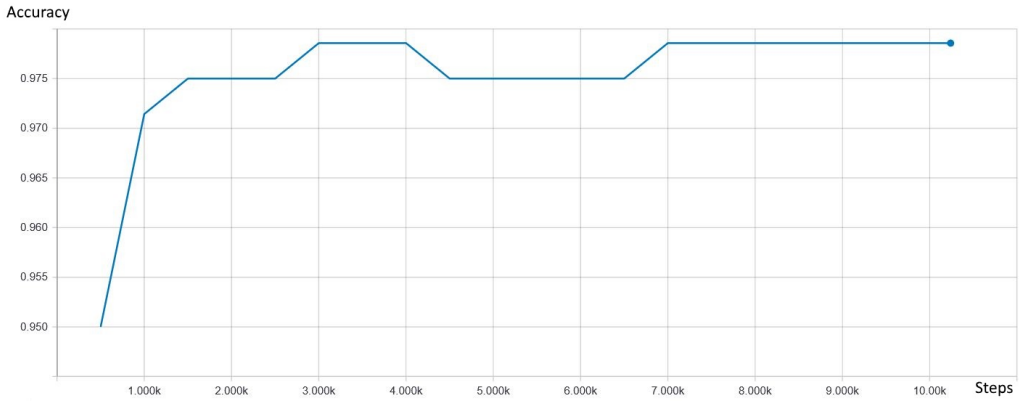

Figure 4.9: Accuracy of the specialized model

5 Convolutional Neural Network Architecture

The convolutional neural network architecture used in this thesis has been originally proposed by Goll [Gol18]. Goll conceptualized and implemented a convolutional neural network for the same rock-paper-scissor classification task considered in this thesis. The architecture makes use of a convolutional and no pooling layer, but uses 3 fully connected dense layers for the classification.

5.1 Structure

Goll used a different sequence length for the input. This is the reason why his input layer consists of a 24×8 input tensor, 24 frames of input sequence across 8 EMG channels. As the proposed sequence length in this thesis is 40 frames, the input layer is adjusted to fit a 40×8 input tensor. The structure is visualized in Figure 5.1.

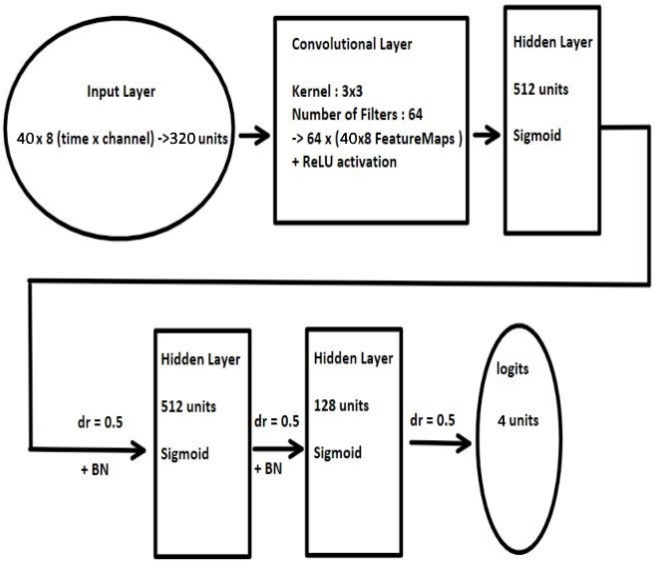

Figure 5.1: Structure of the convolutional neural network (Original taken from [Gol18])

The input layer is then followed by a convolutional layer. 64 filters with a kernel size of 3×3 are applied. Padding is used at the interval limits, ensuring that each feature map is the same size as the input. Applying 64 filters results in 64 feature maps of size 40×8,

The ReLU function is applied as an activation function to the feature maps.

The resulting feature maps are fed into the first dense layer, consisting of 512 hidden units. Between the dense layers, dropout is applied with a dropout probability of 0.5. The second dense layer consists of 512 units and the last dense layer uses 128 units. The final logit layer contains the 3 output nodes which uses the softmax function to obtain the final output values.

5.2 Parameters

The parameters used for the convolutional neural network do not change from the parameters proposed by Goll and are taken from his Bachelor's thesis.

Activation Function The ReLU activation function is used as an activation function for the neural network nodes. The ReLU function is a popular choice, as it counteracts the vanishing gradient problem, promotes sparsity and is comparable easy to compute.

Lossfunction and Regularization The softmax cross entropy function is used as a lossfunction for the model. Regularization in form of L2 or L1 regularization is not applied. However, dropouts are used between the dense layers of the neural network to avoid overfitting. Furthermore, batch normalization is implemented as a means to improve generalization.

Optimizer Goll suggests using the AdagradOptimizer for the model, as it is state of the art for stochastic gradient descent and works well with artificial neural networks.

5.3 Evaluation

The evaluation is done in the same fashion as done for the recurrent neural network architecture. To ensure fairness, the testing and evaluation sets within each experiment are the same.

Test01	Test02	Test03	Highest Accuracy
0.8214	0.8257	0.8157	0.8257

Table 5.1: Evaluation of the highest accuracy

Highest Accuracy The model with the best results achieved a maximum accuracy of 82.57% as seen in Table 5.1. In a similar experiment, Goll reports an accuracy of 73.66%. The results are better than expected. This might be caused by the change of the incoming signal. The original model used input sequences of a 24 frame length compared to the 40 frame input sequence that the model is using now.

How Many Steps until x% Accuracy The original goal was to make the experiments conducted on both the recurrent neural network and the convolutional neural network equal. However as the convolutional neural network never reached 90% accuracy, the metric will be slightly changed to fit the circumstances. The three highest accuracy models' training progress is observed to find out, how many steps it took for each model to reach 80% accuracy. The results can be observed in Table 5.2.

Test01	Test02	Test03	Average steps required
62000	77000	102000	80333

Table 5.2: Training steps required to reach 80% accuracy

An average of 16067 steps was required for the model to reach 80% accuracy on the Top5 dataset. This is relatively slow and could be caused by the large amount of weights caused by using three dense layers for classification plus 64 filters in the convolutional layer.

Complexity The convolutional neural network architecture consists of an input layer, an convolutional layer followed by 3 dense layers. The weights of a convolutional layer is dependent on the number of filters N_{filter}, the filter size $dim_x \times dim_y$. Assuming only one input plane is processed and ignoring any weights related to bias units, the amount of weights in a convolutional layer can be estimated as follows.

$$N_{weights} = N_{filter} \cdot dim_x \cdot dim_y$$

The convolutional layer of the model at hand thus contains about

$$64 \cdot 3\dot{3} = 576 \text{ weights.}$$

The output size is 20480, because 64 feature map of the input size of 40×8 are created. The first dense layer therefore uses

$$20480 \cdot 512 = 10.485.760 \text{ weights.}$$

The following layer uses

$$512 \cdot 512 = 262.144 \text{ weights.}$$

The final layer uses

$$512 \cdot 128 = 65.536 \text{ weights.}$$

The whole neural network uses an estimated 10.814.000 weights, ignoring bias units. It is to note, that convolutional layer adds additional complexity, as each of the 64 filters performs $40 \cdot 8 = 320$ filter operations to compute its feature map.

Computational and Prediction Latency To assess the computational latency, the model was tasked with predicting 1.000 datapoints in quick succession. The experiment

was repeated five times. The resulting times can be seen in Table 5.3.

Test01	Test02	Test03	Test04	Test05	Average time required
15.0947s	15.0479s	14.9854s	14.7822s	15.1574s	15.0135s

Table 5.3: Test results for computational latency in seconds

It took the model around 15.0135 seconds to classify 1.000 training examples. Classifying a single training example therefore takes about 15.01 milliseconds. With this rate, the model can classify about 66 training examples per second, making it useful for real time applications.

As with the recurrent model, the predictive latency was tested using a real time live test. A gesture was inputted simultaneous together with a button press. The time steps needed until the classifier outputted the right classification were counted. Just as the recurrent model, the convolutional model has an predictive latency of 1-3 time steps. Because of the smaller computational latency, the actual delay is between 15.01 and 45.03 milliseconds.

Generalization First, the convolutional model was trained on a subset of the Top5 dataset three times. The best model was chosen from the resulting models. The best model was able to reach a highest accuracy of 80.45% as seen in Figure 5.2.

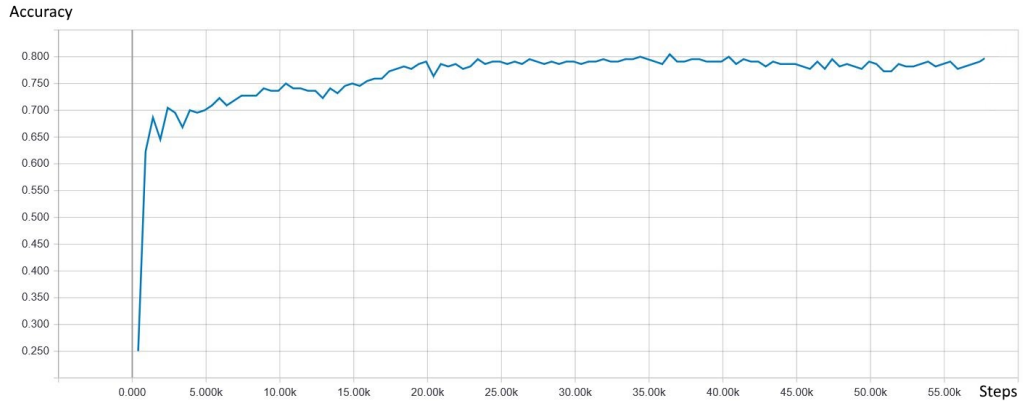

Figure 5.2: Accuracy of the chosen convolutional model

To evaluate the generalization ability, the chosen model was tasked with classifying data from three different persons outside the training set. The model was able to successfully predict 69.16% of the data.

Specialization Three models have been trained on a dataset recorded only by a single person. The accuracies of the resulting models depict how well the neural network architecture can specialize on the data of a single person. The results are found in Table 5.4.

Model 01	Model 02	Model 03	Highest Accuracy
0.8893	0.8893	0.8964	0.8964

Table 5.4: Accuracy of the specialized models

The results are lower than expected with the best model achieving an accuracy of about 90%. Goll reports accuracies up to 99.60% when training the architecture on single users. As slight alterations have been done to the input layer of the architecture, different results were expected. The model seems to specialize worse with the added length of the input sequence.

6 Comparison

A convolutional and a recurrent neural network architecture have been created, trained and evaluated on different metrics. The two architecture will now be compared using the results from the previous experiments. An overview can be found in Table 6.1.

	Recurrent model	Convolutional model
Highest accuracy	0.9371	0.8257
How many steps until x% accuracy	21666 ($x = 90$)	80333 ($x = 80$)
Estimated number of weights	800.000	10.800.000
Computational latency	16.23ms	15.01ms
Prediction latency	1–3 time steps	1–3 time steps
Generalization	0.6791	0.6916
Specialization	0.9786	0.8964

Table 6.1: Evaluation results of both neural network models

Highest Accuracy The recurrent model achieved an accuracy of 93.71% in one of its best runs while the convolutional model only achieved a maximum of 82.57%. It is assumed, that the accuracy of the convolutional neural network could be improved by further testing and adjusting the parameters, since slight alterations have been made to the input sequence compared to the input in Goll's thesis [Gol18].

How Many Steps until x% Accuracy The recurrent neural network architecture trained a lot faster compared to the convolutional neural network, taking only 21666 training steps on average to reach an accuracy of 90%, while the convolutional model needed an average of 80333 training steps to reach 80%.

Complexity The recurrent neural network structure contains about 800.000 weights, distributed across 4 layers. The convolutional structure makes use of 5 layers with a total number of 10.800.000 weights. The memory requirements for the convolutional model are thus higher. As revealed in the next paragraph, the higher number of weights does not necessarily mean that the convolutional model is slower to compute.

Computational and Prediction Latency To compute one prediction, the recurrent model takes about 16.23 milliseconds. The convolutional model needs about 15.01 milliseconds for one prediction, making the model about 7.5% faster. This is surprising, as the convolutional model has about ten million more weights compared to the recurrent model.

It is assumed, that the difference lies in the way a recurrent layer computes its output. The data of each time step needs to be processed sequentially, as the resulting cell states after the first time step affect the calculations of the second time step and so forth. The convolutional layer however, is able to process the whole data sequence in parallel, thus being able of take advantage of multithreading.

The prediction latency is the same for both models. Each model takes between one and thee time steps to recognize and classify a performed gesture. The convolutional model can process about 66 data sequences each second, making it the more responsive model compared to the recurrent model, which can process about 60 data sequences each second.

Generalization Both models generalize quite poorly with accuracies of 67.91% for the recurrent- and 69.16% accuracy for the convolutional model. The overall low accuracies in this category could be due to the sEMG data being very unique from person to person, making generalization difficult. For better generalization results, it is assumed that a bigger dataset from more different people would probably need to be recorded.

Specialization By training the networks on data from only one source, the models' ability to specialize was revealed. The recurrent model achieved 97.86% in this experiment, making it the better specializing model when compared to the 89.64% accuracy achieved by the convolutional model.

7 Conclusion

7.1 Related Work

Chowdhury, M. B. Reaz, Ali, et al. [Cho+13] give an in-depth overview on sEMG signal processing and classification techniques. Similar work has been done by M. Reaz, Hussain, and Mohd-Yasin [RHM06], illustrating various EMG signal analysis methodologies. Deep learning approaches for the classification of sEMG data have also been examined by Du, Jin, Hu, and Geng [Du+17]. A review about sEMG signal classification for the purpose of human computer interaction has been done by Ahsan, Ibrahimy, and Khalifa [AIK09]. A fighting game that is controlled using sEMG signals and acceleration data has been implemented by Park and Kim [PK11].

7.2 Summary and Future Work

The goal of this thesis was twofold. The first goal was to investigate the usefulness of EMG data for gesture classification using artificial neural networks. The second goal was to provide a comparison between a recurrent- and a convolutional neural network architectures, after they have been evaluated on different metrics. On a dataset containing 2000 gestures recorded by 5 different persons, the recurrent neural network model achieved a 93.71% accuracy, while the convolutional model achieved an accuracy of 82.57%. Since it was unavoidable to introduce noise to the dataset due to human error, it is expected that some examples are wrongly classified. A real time test indicated that the error can be further reduced by using a complementary postprocessing step which returns the most frequent classified gesture of the past x time steps. EMG techniques are thus a promising candidate for hand gesture classification.

A comparison between the two investigated neural networks has been made in Chapter 6. When it comes to accuracy, the recurrent model trumps over the convolutional model achieving a 93.71% accuracy compared to the 82.57% achieved by the convolutional model. The average steps required to reach that level of accuracy are much lower in the case of the recurrent model, with only about 20.000 steps required compared to the 80.000 steps, which the convolutional model needs. The number of weights play a big part in this, as the recurrent model uses only 800.000 weights. The convolutional model on the other hand, takes use of over 10.000.000 weights, which explains the amount of training steps required to reach acceptable accuracies. Even though the number of weights is much higher, the convolutional model is faster when it comes to computation time with an average computation time of 15.01 milliseconds. The recurrent model has a computation time of 16.26 milliseconds. Since both models have a latency when making predictions of roughly 1-3 time steps, the lower computational latency make the convolutional model

more responsive, however the difference remains relatively small. Both models performed roughly the same in the generalization test, with a 67.91% accuracy of the recurrent- and a 69.16% accuracy of the convolutional model. The difference in specialization capabilities is much bigger however. The recurrent model achieved an accuracy of 97.86% in the specialization test, while the convolutional model only achieved 89.64%.

To summarize, in almost all aspects of accuracy, the recurrent model is advantageous compared to the convolutional model, however both models generalize equally well. On the other hand the specialization ability of the recurrent model is better. The memory usage of the convolutional model is much higher due to the much higher number of weights. The computational latency introduced by the complexity of both neural network models is roughly equal, with the convolutional model being about 7.5% faster. There is no noticeable difference when it comes to prediction latency.

For these reasons, the recurrent model is deemed to be more suitable for the task of gesture recognition, since it is easier to train, requires less memory and achieves better accuracy, while being only slightly slower to compute.

During this thesis, some ideas for improvement came to mind, which could not be pursued due to the limited time available. Both model's accuracy and especially generalization ability could be improved by recording a bigger dataset from different persons. It is assumed, that the parameters and structure of the convolutional neural netwok can be altered to achieve better results, as the current architecture has not been developed using the same kind of input as used in this thesis. During development, attempts have been made to create a combined model, making use of first the convolutional layer and then the two recurrent layers of both models. While the model returned slightly worse results compared to the recurrent neural network, it is believed, that with further testing the mixed model might outperform both models presented in this thesis.

List of Figures

List of Tables

Bibliography

[AH17] H. H. Aghdam and E. J. Heravi. *Guide to Convolutional Neural Networks : A Practical Application to Traffic-Sign Detection and Classification*. Springer, 2017.

[AIK09] M. Ahsan, M. I. Ibrahimy, and O. O. Khalifa. "EMG Signal Classification for Human Computer Interaction: A Review." In: *European journal of scientific research* 33.3 (2009).

[App17] C. V. M. L. T. (Apple). *An On-device Deep Neural Network for Face Detection*. 2017. URL: https://machinelearning.apple.com/2017/11/16/face-detection.html (visited on 08/27/2018).

[Bud16] A. Budhiraja. *Dropout in (Deep) Machine learning*. 2016. URL: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5 (visited on 09/04/2018).

[Cho+13] R. H. Chowdhury, M. B. Reaz, M. A. B. M. Ali, A. A. Bakar, K. Chellappan, and T. G. Chang. "Surface Electromyography Signal Processing and Classification Techniques." In: *Sensors* 9 (2013).

[Dal18] N. Dale. *Deep learning for autonomous cars*. 2018. URL: https://verneglobal.com/blog/deep-learning-for-autonomous-cars (visited on 08/27/2018).

[DHS11] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159.

[Dia17] E. Diaz-Aviles. *A Glimpse into Deep Learning for Recommender Systems*. 2017. URL: https://medium.com/@libreai/a-glimpse-into-deep-learning-for-recommender-systems-d66ae0681775 (visited on 08/27/2018).

[Du+17] Y. Du, W. Jin, Y. Hu, and W. Geng. "Surface EMG-Based Inter-Session Gesture Recognition Enhanced by Deep Domain Adaptation." In: *sensors* 3 (2017).

[FBB11] X. Florot, A. Bordes, and Y. Bengio. "Deep Sparse Rectifier Neural Networks." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011).

[Gol18] L. Goll. "An Adaptive Interface for Individual Surface EMG Classification Using an Artificial Neural Network." Bachelor Thesis. München: Technische Universität München, 2018.

[Gon15] N. Gonzalez. *Use Hand Gestures to Take Selfies More Easily on Android*. 2015. URL: https://android.gadgethacks.com/how-to/use-hand-gestures-take-selfies-more-easily-android-0160677/ (visited on 08/27/2018).

[Goo] Google. *Tensorflow*. URL: https://www.tensorflow.org/ (visited on 09/04/2018).

[Hin] G. Hinton. *Overview of mini-batch gradient descent*. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visited on 09/04/2018).

[HS97] S. Hochreiter and J. Schmidhuber. "Long Short-term Memory." In: *Neural Computing* 9 (1997), pp. 1735–1780.

[Hum17] R. Hume. *From the handshake to the high-five: a brief history of gestures*. 2017. URL: https://www.historyextra.com/period/ancient-greece/a-brief-history-of-gestures-from-the-handshake-to-the-high-five/ (visited on 08/27/2018).

[KB17] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *3rd International Conference for Learning Representations*. 2017.

[Kon18] A. Konar. *Gesture Recognition*. Springer International Publishing, 2018.

[LCB] Y. LeCun, C. Cortes, and C. J. Burges. *The MNIST Database of handwritten digits*. URL: http://yann.lecun.com/exdb/mnist/ (visited on 08/27/2018).

[OR16] F. J. Ordóñez and D. Roggen. "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition." In: *Sensors* 1 (2016).

[Pan+14] V. V. Pande, N. S. Ubale, D. P. Masurkar, N. R. Ingole, and P. P. Mane. "Hand Gesture Based Wheelchair Movement Control for Disabled Person Using MEMS." In: *Int. Journal of Engineering Research and Applications* 4 (2014), pp. 152–158.

[PK11] D. G. Park and H. C. Kim. "Muscleman: Wireless input device for a fighting action game based on the EMG signal and acceleration of the human forearm." In: (2011).

[RHM06] M. Reaz, M. Hussain, and F. Mohd-Yasin. "Techniques of EMG signal analysis: detection, processing, classification and applications." In: *Biol.Proced.Online* 8(1) (2006).

[Sam18] V. Sampath. *Convolutional Neural Networks for Autonomous Cars*. 2018. URL: https://electronicsforu.com/market-verticals/automotive/convolutional-neural-networks-autonomous-cars-part-1-2/3 (visited on 09/04/2018).

[Spi15] S. Spieckermann. "Multi-Task and Transfer Learning with Recurrent Neural Networks." Dissertation. Muenchen: Technische Universität München, 2015.

[SS15] P. K. Sharma and S. Sharma. "Evolution of Hand Gesture Recognition : A Review." In: *International Journal Of Engineering And Computer Science* 4 (2015), pp. 9962–9965.

[Tak12] H. Takada. *Electromyography : New Developments, Procedures and Applications.* Nova Biomedical, 2012.

[You+18] T. Young, D. Hazarika, S. Poria, and E. Cambria. "Recent Trends in Deep Learning Based Natural Language Processing." In: (2018).

[Zei12] M. D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method." In: (2012).