DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Classification of sEMG using TensorflowJS in Ubi-Interact - Playing Rock-Paper-Scissors

Anastasia Pomelova

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Classification of sEMG using TensorflowJS in Ubi-Interact - Playing Rock-Paper-Scissors

Klassifizierung von sEMG mithilfe von TensorflowJS in Ubi-Interact - ein Schere-Stein-Papier-Spiel

Author:	Anastasia Pomelova
Supervisor:	Gudrun Klinker
Advisor:	Sandro Weber
Submission Date:	15.09.19

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, 15.09.19 Anastasia Pomelova

Acknowledgments

I would like to thank Prof. Gudrun Klinker and Sandro Weber for accepting my bachelor thesis topic and for providing me with materials and aid during the process. Big thanks to Klaus for tolerating my complaints, because as my flatmate they are hard to escape, and Julius for keeping me sane with obscure memes. Another thank you to my dad and all my other friends for being patient with me and everyone who still believes that one day I will actually dungeon master all the rpg-campaigns that I promised when I was stressed out.

Abstract

Gestures can be an interesting input method for various applications. There are several approaches on how to detect them, as new sensors provide different kinds of data to work with. One kind is muscle-specific data received via surface electromyography. It can be used to classify hand-gestures recorded by wearable sensors. In this thesis, the Myo armband and machine learning techniques are used to detect these gestures to play a game of Rock-Paper-Scissors. An interface for collecting sEMG data was developed in the Ubi-Interact framework. For gesture classification an existing Tensorflow model was trained and integrated into the JavaScript environment using Tensorflow.js. In the end, it was possible to build an application to play Rock-Paper-Scissors against a virtual opponent using hang-gesture input.

Contents

1 Introduction

1.1 Motivation and Background

1.1.1 Gesture Recognition

Gestures are a form of non-verbal communication that carries a certain message or meaning and involve body movement of arms, legs or muscles of the face. With gesture recognition techniques, we can access that meaning and use it to build human-computer interaction [Kon18].

Our hands are the main mean to physically interact with our surroundings. Therefore hand-gesture recognition is interesting for different applications like virtual reality. It enhances the interaction experience and integrates the real world into the virtual one. With it, we can create a more realistic and immersive interaction compared to traditional input methods. [RA11]

Another point is the spatial consideration: gestures can eliminate the need for physical touch as input. This enables interaction from a distance [RA11] which can be beneficial in several kinds of applications:

In a surgery setting a touchless interface can reduce time and potential infection risks [Nas+16]. But even in our day-to-day life can a gesture-based interface be beneficial. For example, accepting a call in our car [Joh+17] or taking a picture with a simple hand-gesture instead of setting a timer [Sam18].

Visual sensors can be used for hand-gesture detection like the commercially available Leap Motion[1] and the Microsoft Kinect[2]. But the position of those sensors has to be considered. For example, the Leap Motion can be placed in front of the user or be attached to a VR-headset. This placement influences the tracking quality of the sensor. The gesture has to be performed towards it for best results.[ZM17] If the fingers are covered by the back of the hand detection gets more difficult. Therefore for simple pointing gestures, the Leap Motion performs better when placed in front of the user. The disadvantage is that the user is bound to a specific area unlike the head-mounted approach.

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

[2] https://developer.microsoft.com/en-us/windows/kinect

Another option are wearable technologies for gesture recognition. Those sensors are placed directly on the body. Data gloves [BeB19] can determine the relative position of the fingers but are more restricting.

What we will investigate in this thesis is the Thalmic Labs Myo armband [Lab18]. It can read muscle contractions via surface electromyography (sEMG). As the Myo does not provide us with the exact positions of the fingers we will use machine learning techniques to classify the sEMG data. This way we can detect hand-gestures without restricting the hand itself or depending on lighting conditions or a certain position of a visual sensor.

1.1.2 Rock-Paper-Scissors

Rock-Paper-Scissors is a simple hand-gesture based game. It is known by many people or is at least easy to learn as it only consists of the three name-giving gestures:
rock (fist), paper (flat hand) and scissors (two fingers stretched out). All of them are easily distinguishable with the human eye. This leads to the question: how well can a machine distinguish these three gestures? An implementation of the game would also benefit from hand-gesture input as it is the natural way to play that game. So we will use a Rock-Paper-Scissors game to investigate the possibilities of gesture classification using sEMG data.

1.2 Related work

There have been several approaches to use the Myo armband as input for various human-computer interaction tasks. From robotics [HAI19] and prosthetic hand control [Chu+17] to musical expression [NHJ15] and map navigation [SM15].

1.2.1 Myo sEMG classification

We are interested in classifying Myo's sEMG data with machine learning. This also has been done before: Goll [Gol18] build a convolutional neural network to classify Myo gestures. Another approach from Fromm [Fro18] is using a recurrent neural network. Both networks have been implemented using Tensorflow [Aba+16] and trained to classify the gestures needed to play Rock-Paper-Scissors. Another neural network for classifying a total of five gestures was build by Mytrovtsiy [Myt18].

Instead of building an own model from scratch, we are going to load one of the existing models and investigate how well it can be integrated into another application to perform gesture recognitions tasks.

1.3 Goal

The goal of this thesis is to build an application where we can play Rock-Paper-Scissors using hand-gesture input. To achieve it, we collect sEMG data with the Myo armband and classify this data using a pre-trained neural network. For accessibility and reusability reasons we are going to use a web-based approach: We will implement a Myo interface and the final game in the Ubi-Interact framework.

2 Theory

2.1 Electromyography

Electromyography is recording the electrical activity of a muscle. It can be used for clinical examination like detecting abnormalities, muscle wasting, and weakness [Mil05]. But it can also be used for human-machine interaction in the form of gesture recognition [Zha+11] and virtual reality applications [PK].

We distinguish between two categories of EMG: Intramuscular and surface electromyography (iEMG and sEMG). In iEMG a concentric needle electrode is inserted into the muscle for recording [Mil05]. The non-invasive method is sEMG. Here the electrode is placed on the skin above the area of interest. Both methods, but especially sEMG are prone to noise as the signal travels through different tissues and EMG detectors may collect signals from different motor units [RHM06].

In conclusion, it is a complex signal, controlled by the nervous system and dependent on physical properties of the muscle [RHM06]. Difficulties arise because of the stochastic nature and noise of the signal. Hence a machine learning approach is useful for interpreting it [Mit16].

2.2 Machine learning

"Machine learning is based on algorithms that can learn from data without relying on rules-based programming" [PJ15]. Since its beginnings in the 1990s, it advanced to an important technical field due to digitalization and cheap computing power [PJ15]. It is gaining interest as machine learning systems are highly adaptable and improve through experience [JM15]. As of today, machine learning is used in a variety of applications and areas. For example, in the banking sector [Son19], for face recognition [Hig15], or playing complex games like chess or go [Sil+17b].

2.2.1 Pattern recognition problems

Solving a pattern recognition problem means detecting a recurring pattern in the given data to classify it. Data in the same class have similar properties [Fu12]. This method can be used for tasks like speech recognition, image detection, and gesture classification.

2.2.2 Artificial Neural Networks

Artificial neural networks can solve pattern recognition tasks. They are inspired by human brain cells and consist of several layers of neurons, where each layer is connected to the next one [Sil+17a]. The basic architecture of an artificial neural network includes an input layer, an output layer and a variable amount of hidden layers. The input layer's dimension corresponds to the input data. The same applies to the output layer and the output data [Sil+17a].

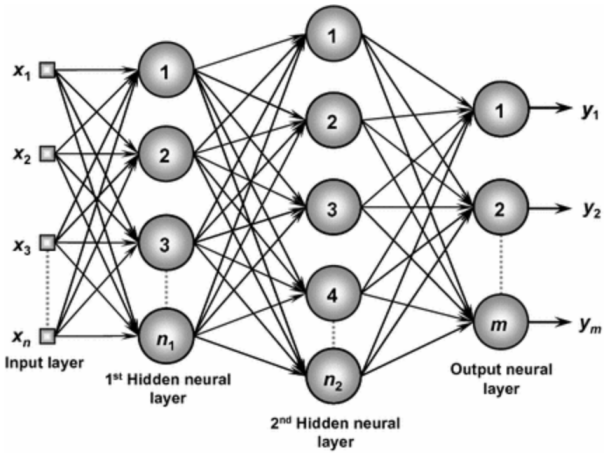

Figure 2.1: An example of a artificial neural network with multiple layers [Sil+17a]

Each neuron takes input from the previous layer, performs a calculation on it and outputs a new value [Nie15]. The calculation can be described as follows:

$$a(\sum_{i=0}^{n}(x_i * w_i) + b)$$

Each input value x_i is multiplied with a weight w_i according to its importance and summed up adding a bias b for scaling. a is the activation function of the neuron used to normalize the output [Nie15]. The values of the weights and biases are determined during the training process.

3 Fundamentals

3.1 Hardware

3.1.1 Myo armband

The Myo armband is a commercially available sensor distributed by Thalmic Labs between 2013 and 2018 [Lab18]. It has eight stainless steel sEMG sensors on the inside of the armband. Moreover, it has a nine-axis inertial measurement unit (IMU) with a three-axis gyroscope, accelerometer, magnetometer and an ARM Cortex M4 processor. It can be connected via Bluetooth to Windows, Mac, iOS and Android devices. The Myo can detect 5 pre-defined gestures: fist, double-tap, fingers-spread, swipe-in, and swipe-out. It also gives us access to the raw sEMG data for our own gesture classification approaches. The armband is placed on widest part of the forearm for best detection.

Figure 3.1: Thalmic Labs Myo armband

3.2 Software

3.2.1 Ubi-interact

"Ubi-Interact is a framework for building reactive and distributed applications" [19]
It brings different devices together, which are connected to a server as clients. These clients can send their data via network and publish it to topics. The server redistributes this data to all devices who subscribed to that topic. It can also process the data in interactions. For example, an interaction can perform gesture recognition tasks and return the performed gesture. Ubi-Interact is dependent on the data formats and not the actual devices, which makes "devices exchangeable" and "interactions reusable" [19].

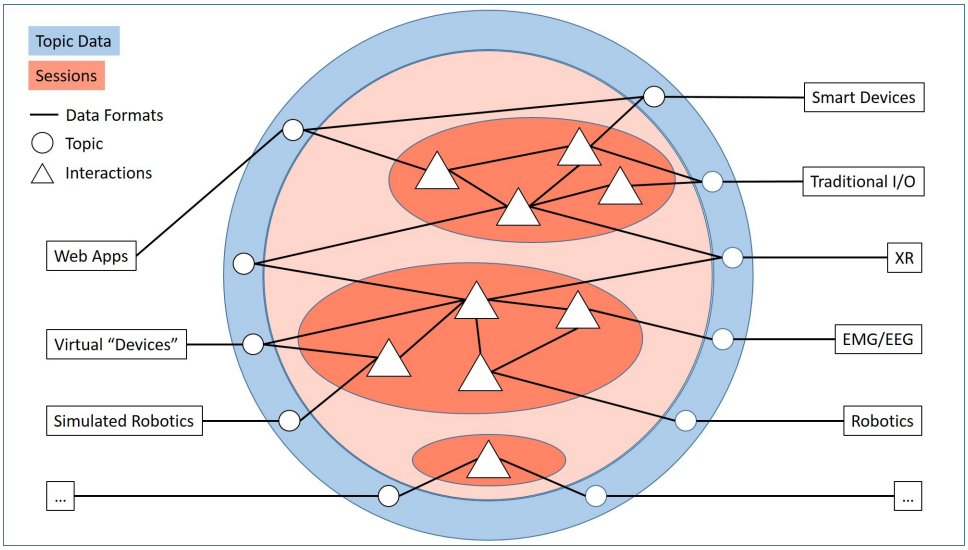

Figure 3.2: Overview Ubi-Interact [19]

The backend is built with the JavaScript runtime Node.js[1] and the frontend is realized in Vue.js[2]. The sent data is defined using Google Protocol Buffers[3].

3.2.2 Myo.js

Myo.js[4] is a JavaScript framework for accessing Myo data in the browser or Node.js.

3.2.3 Tensorflow

Tensorflow is an open-source library for developing and training machine learning models [Aba+16]. Its core is written in C++ with frontend-bindings in C++ and Python.

[1]https://nodejs.org
[2]https://vuejs.org
[3]https://developers.google.com/protocol-buffers
[4]https://github.com/thalmiclabs/myo.js

[Sch16]. Since Tensorflow went open-source in November 2015 [Aba+16] it became the most popular deep learning framework [Hal18].

Tensorflow.js

Tensorflow.js is a library for developing machine learning models in JavaScript. It can run on client and server-side and is compatible with the Tensorflow API, so models can be ported between Python and JavaScript [Smi+19]. This is especially interesting since, according to the Stack Overflow Developer Survey 2019, JavaScript is the most commonly used programming language [Ove19]. Machine learning libraries on the other side are often written for Python or C++ developers [Tec18]. This is rooted in speed reasons and accessibility to the GPU [Smi+19]. Tensorflow.js repurposes the web platform's graphics API for high-performance computation and is the first machine learning platform to enable integrated training and inference on the GPU from the browser. Besides that, it offers full Node.js integration. This makes it compatible with numerous devices [Smi+19].
Furthermore offers Tensorflow.js already pre-trained models[5] for tasks like image classification[6].

Tenserflow.js converter

Tensorflow.js offers a model converter for Tensorflow GraphDef based models. It outputs the model structure as a `model.json` file and its weights as binary files [Tenb]. We can access the converter by installing the Tensorflow.js pip package. The original model has to be saved independently of the code that generated it. Convertible formats are the SavedModel, FrozenModel, SessionBundle and Tensorflow Hub module [Tena]. We are using a Saved Model for conversion [Tenc]. Here the model is saved as a `saved_model.pbtx` protocol buffer file. It contains the graph structure, with an additional variables folder for the learned weights.

3.2.4 EMG classifier model

The model for the gesture recognition task is a neural network able to classify Myo sEMG data written in Tensorflow by Mytrovtsiy [Myt18]. The network takes 8 consecutive sEMG samples as input. Each sample has 8 values, one for each sEMG sensor of the Myo armband. So, in total, the input consists of 64 numeric values. The network has an input layer, an output layer, and two hidden layers and uses an rectified linear unit activation function.

[5] https://github.com/tensorflow/tfjs-models
[6] https://github.com/tensorflow/tfjs-models/tree/master/mobilenet

4 Concept and Design

4.1 Idea

Our goal is to play a game of Rock-Paper-Scissors using the Myo armband. At first, we need to figure out our subtasks and decide on a design for the whole application.
We can break down our task into three logical steps:

1. getting the Myo data
2. classifying the data to receive the performed gesture
3. playing Rock-Paper-Scissors with said gesture as input

We have to handle data acquisition, data interpretation, and the actual gameplay. With our subtasks defined we can start making design decisions.

4.2 Design

The design has to fit our working-environment, which is Ubi-Interact. For example, we want the components to be reusable. Therefore, data acquisition and data usage have to be separated. A Myo interface should handle the input and a Rock-Paper-Scissors application handles the gameplay. So the Myo data could be used for other purposes.

The interface collects data and publishes it to the Myo data topic. The game application creates a session which takes that topic as input. It collects the data until it has 8 sEMG-samples and classifies it on server-side with a Tensorflow.js model. The output of this interaction is a classified gesture, represented as a number, which will be used as input for the Rock-Paper-Scissors game.

Figure 4.1 shows the design for our system and illustrates the connection between the Myo interface, the game, and the backend.

As we need communication between our different system parts, we have to consider message formats for our topic data: We need representation for the Myo data and an integer number. Ubi-Interact has already fitting message formats for standard data like numbers and strings. But to represent all the relevant information from the Myo armband, we will need to define a new message format.

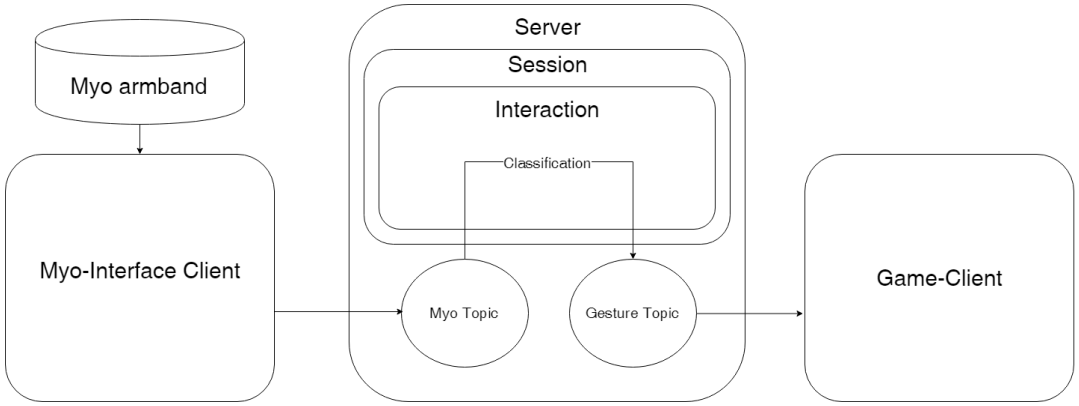

Figure 4.1: System overview

Further, we need to classify the sEMG data by loading a pre-trained Tensorflow.js model in the Ubi-Interact backend. This requires handling conversion and training of a Tensorflow model, which will happen outside of the Ubi-Interact environment.

4.3 Process

After figuring out the basic design for our task, we will formulate the outline for implementation.

1. create Myo Protocol Buffer message format
2. implement Myo interface
3. convert Tensorflow model to Tensorflow.js and load it in Ubi-Interact
   - train model with own data
   - save Tensorflow model code independently
   - convert saved model to Tensorflow.js model
   - make a npm package and include it in the Ubi-Interact backend
4. implement core Rock-Paper-Scissors game
5. connect game application to model and Myo interface

The numbered points were part of the original plan, the sub-points are additions that had to be made during development. All these steps will be further discussed in the following Implementation chapter.

5 Implementation

5.1 Protocol Buffer messages

At first, we define a new message format for the Myo data. We need the sEMG data for gesture classification, which consists of 8 values from 8 sensors. But the Myo has more interesting data: IMU (magnetometer, gyroscope, accelerometer) and gesture. Since the goal is to make the interface reusable, we created a data format containing everything mentioned above.

```
syntax = "proto3";
package ubii.dataStructure;

import "proto/topicData/topicDataRecord/dataStructure/vector8.proto";
import "proto/topicData/topicDataRecord/dataStructure/vector3.proto";
import "proto/topicData/topicDataRecord/dataStructure/quaternion.proto";
import "proto/topicData/topicDataRecord/dataStructure/handGestureType.proto";

message MyoEvent {
  ubii.dataStructure.Vector8 emg = 1;
  ubii.dataStructure.Quaternion orientation = 2;
  ubii.dataStructure.Vector3 gyroscope = 3;
  ubii.dataStructure.Vector3 accelerometer = 4;
  ubii.dataStructure.HandgestureType gesture = 5;
}
```

Code Listing 5.1: Myo message format

The first four fields are vectors of length three to eight and the HandGestureType is an enum containing all possible gestures and a rest gesture when no specific gesture is detected.

5.2 Myo interface

After we defined the message format, we can fill our Myo topic with data. Therefore, we need to set up the interface, collect data and then publish it. At first, we integrate our interface into Ubi-Interact by registering it as a new device. We specify it with a name, type, and its components, stating all input and output formats. The device only has an

```
syntax = "proto3";
package ubii.dataStructure;

enum HandGestureType {
  REST = 0;
  FINGERS_SPREAD = 1;
  WAVE_IN = 2;
  WAVE_OUT = 3;
  FIST = 4;
  DOUBLE_TAP = 5;
}
```

Code Listing 5.2: Hand-gesture enum

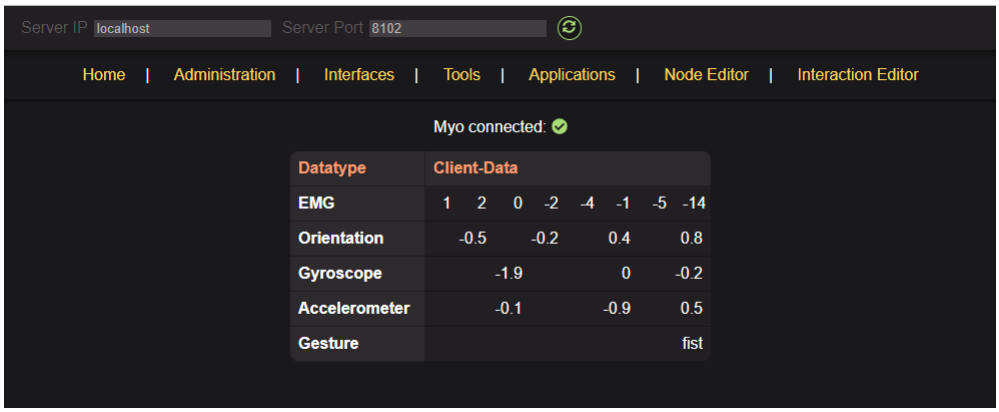

Figure 5.1: Myo interface UI

input, as it only publishes data to the backend. Next, we need to get Myo data with Myo.js by setting up listeners for IMU, sEMG and gesture data. For sEMG data we have to enable the sEMG stream. Unfortunately, this makes Myo's own gesture detection worse. This could be enabled/disabled in the future if a specific application needs the predefined gesture set and wants a more precise detection. For now, we prioritize getting the sEMG stream. Finally the collected data is displayed in the interface and published to the Myo topic in a 10 milliseconds interval.

5.3 Tensorflow model conversion and training

As explained in the Concept and Design chapter before, for converting a Tensorflow model into a Tensorflow.js model it needs to be saved in a code independent format. Unfortunately, this was not the case for any of the Tensorflow models handling Myo gesture classification available to us. Only the weights were saved and not the whole graph structure. This means retraining and correctly exporting the model first.

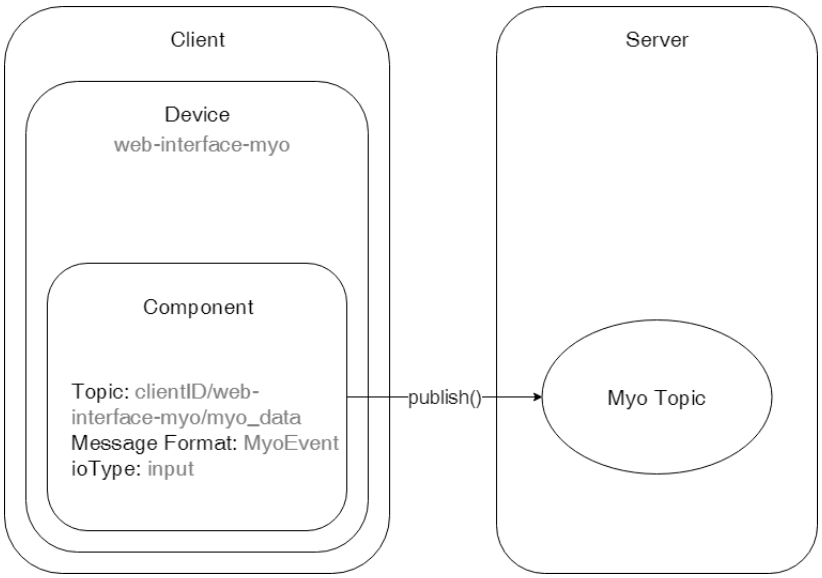

Figure 5.2: Myo interface overview

5.3.1 Myo poller

Before we can start training the model we need data as input for it. Therefore, we need an sEMG data poller that exports our recorded training data as a CSV-file. As we are already working with Myo.js we use it to write a Myo poller script. It records data on button press and labels it according to the selected gesture. For our model, we need a format of 8x8 sEMG samples and an integer value as gesture id.

5.3.2 Exporting the Model

SaveModel: method 1

The Tensorflow model needs to be saved code-independently in a SavedModel format for further use. For that, we define a SavedModelBuilder that will build a SavedModel protocol buffer[1]. The SavedModelBuilder requires some information about the model. For saving we define a SegnatureDef protocol buffer. We need to get the first and last tensor of the model and pass them to the SignatureDef builder by creating a TensorInfo protocol buffer objects that encapsulate the input and output tensors. We pass these to the SaveModel builder module. After that we add the meta graph and variables to the SavedModelBuilder, including a reference to the current session, because we need the weights and our defined prediction signature for building the graph. We can use this method on an already trained network by loading the trained weights. In our case we export the SavedModel when we finish training it.

[1]https://www.tensorflow.org/api_docs/python/tf/saved_model/Builder

```
""" imports and model definition """

#define SavedModelBuilder including path for saving the model
builder =tf.saved_model.builder.SavedModelBuilder(SAVE_PATH)

#create TensorInfo protocol buffer of input and output tensors
tensor_info_input =tf.saved_model.utils.build_tensor_info(input_tensor)
tensor_info_output =tf.saved_model.utils.build_tensor_info(output_tensor)

#define signature
prediction_signature =(
  tf.saved_model.signature_def_utils.build_signature_def(
    inputs={'8xEMG data':tensor_info_input},
    outputs={'gesture':tensor_info_output},
    method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME
  )
)
#add current meta graph
builder.add_meta_graph_and_variables(
  sess,
  [tf.saved_model.tag_constants.SERVING],
  signature_def_map={
  'predict_gesture':
  prediction_signature,
  }
)
def train(num_iterations):
  for i in range(num_iterations):
    """ training """

  #export SavedModel
  builder.save(as_text=True)
```

Code Listing 5.3: Export Tensorflow SavedModel by building it manually

SaveModel: method 2

After the classification abilities of the first training set were not satisfying (see Chapter Training), we retrained the model and investigated the code once again. With a better understanding of Tensorflow, we could refactor the SaveModel export. Instead of manually building a SavedModel, we used the simple save method. For this, we only need to pass it the session, the input tensor, and the output tensor. The rest is being defined automatically. This is shown in Code Listing 5.4

```
""" imports and model definition """

def train(num_iterations):
  for i in range(num_iterations):
    """ training """

  #export SavedModel
  tf.saved_model.simple_save(
    sess,
    SAVE_PATH,
    inputs={'8xEMG data':input_tensor},
    outputs={'gesture':output_tensor})
```

Code Listing 5.4: Export Tensorflow SavedModel with simple save

5.3.3 Model conversion

After training and exporting our model as a SavedModel, we can convert it now with the Tensorflow.js converter by passing it the model with the following command:

```
tensorflowjs_converter
        --input_format=tf_saved_model
        --output_format=tfjs_graph_model
        --signature_name=predict_gesture
        --saved_model_tags=serve
        /path/to/saved_model
        /export/path
```

Code Listing 5.5: Converting the SavedModel into a Tensorflow.js model

When using the simple save method from SaveModel: method 2 the `signature_name` has to be changed to the default signature name `serving_default`. Otherwise, we choose the name we defined ourself.

Training

Unfortunately, the neural network wasn't well documented. The data format for the training data turned out not to be not a CSV-file but a NpzFile. Therefore an data converter script had to be included.

After the data acquisition process, which will be discussed in the Training chapter, the network was trained for 50,000 iterations.

5.3.4 Npm package

The next step is integrating the converted model in Ubi-Interact. For reusability reasons we made a npm package that will load our model, similar to official Tensorflow.js models[2]. This way, we can update it outside of Ubi-Interact. The model itself is hosted on a university server and is loaded via URL.

5.4 Rock-Paper-Scissors application

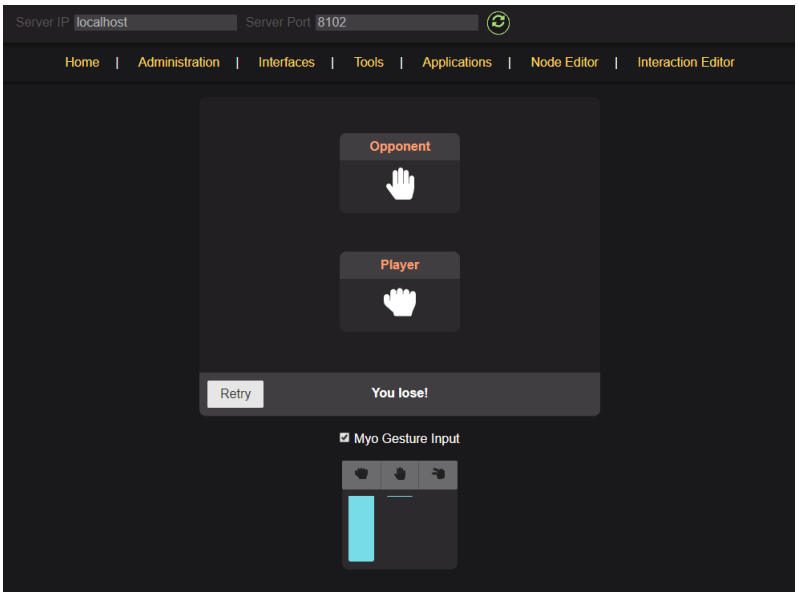

Figure 5.3: Rock paper scissors game UI

5.4.1 Gameplay

Finally it is time to handle the Rock-Paper-Scissors game. We want the player to be able to select the gesture via button or perform it directly using the Myo armband and the Myo interface. The input method can be selected via check-box, with gesture input as default. One game has the following sequence:

- player presses the ready button
- countdown
- user input is registered during a two-second interval
- visual feedback for the player

[2]https://github.com/tensorflow/tfjs-models

During the input interval, an array is filled with classified gestures. The percentages of the currently detected gestures are displayed as a graph under the input buttons for feedback. The most frequent gesture is taken as the player input. After that, a random gesture for the opponent is selected and the game evaluated. The two gestures and a fitting win, lose or draw message is displayed. The evaluation is based on the following rules: rock beats scissors, paper beats rock, scissors beats paper and two identical gestures are a draw. This feedback screen can be found in figure 5.3.

5.4.2 Gesture classification

Similar to the Myo interface, we start by defining a new device and connecting the application to the Ubi-Interact backend. The whole data handling and classification process should happen in an interaction on the server-side. Therefore, we want to define a session and its interaction. The session gets the data from the Myo interface as input and returns a classified gesture as a new topic. The device subscribes to that topic and uses this data for the game.

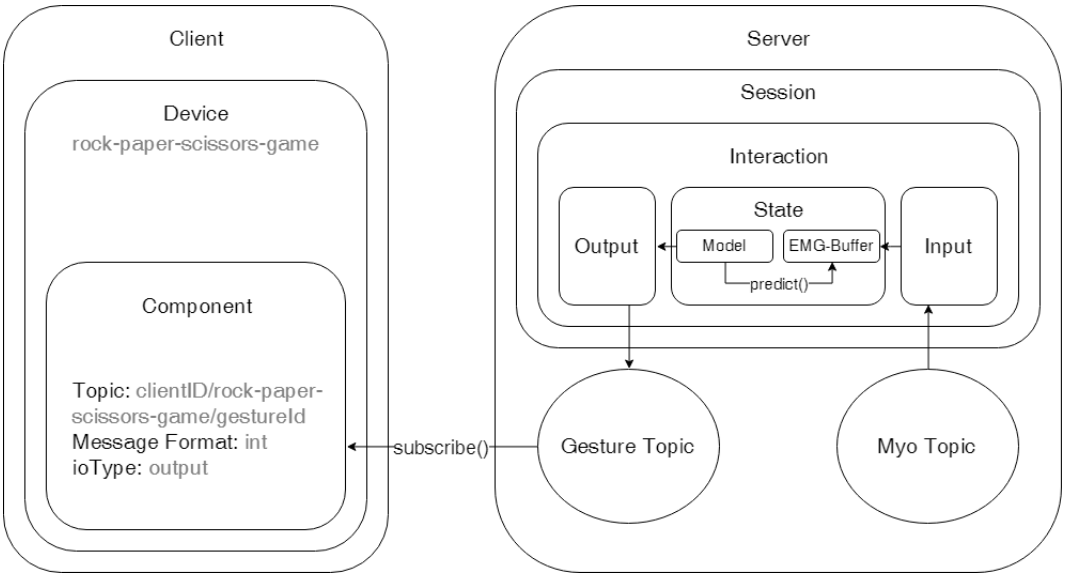

Figure 5.4: Gesture classification in interaction

In contrary to the Myo interface, we only have an output component for the device. We handle the classification in the interaction, which gets its input from the Myo topic. To access the topic we need its full name, including device and client id. It can be found by iterating through the list of all topics and searching for one that contains `myo_data`. The classification is handled by the interaction in an on-create and a callback function. The on-create should set up everything for the classification process. Each interaction has a state that we use for caching data. First, we load the sEMG classifier model into the state. We already imported the emgClassifier package in the backend, so we can

access it via state.module and asynchronously load the model into the interaction state. Furthermore, we define an sEMG-buffer for later use. In the callback-function, we need to handle two things. We are collecting 8 sEMG samples in the sEMG-buffer array and classify them with our model when we have enough samples. These samples are saved in the interaction-state. The model returns the classified gesture and updates the gestureID topic. The Rock-Paper-Scissors application itself subscribes to that topic and receives new data on every classification.

6 Training

6.1 Gesture set

We define the exact sensor position for training: The Myo armband is placed on the widest part of the forearm, the logo should be under the elbow pit, facing away from the body. For the hand-gestures, we define the following:

- **default**: muscles are relaxed, finger facing down
- **rock**: closed fist, thumb visible
- **paper**: hand stretched out, palm faces down
- **scissors**: index and middle finger stretched out in a "v" position, other finger closed

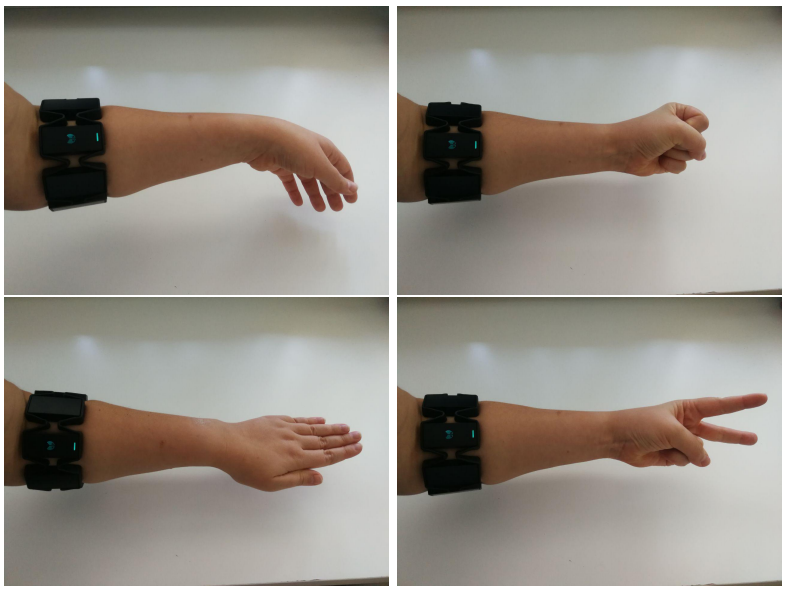

Figure 6.1: Trainings gestures and sensor position. (top-left: default, top-right: rock, bottom-left: paper, bottom-right: scissors)

We recorded different training sets from one user. The first ones had around 5,000 samples and were used to train the neural network for 10,000 iterations. Unfortunately, the model did not reach a classification accuracy over 60% with this trainings data. In

the final game, every gesture except rock did not feel well recognized. We decided to investigate how we can improve our training data and with it the model for overall better classification.

6.2 Polling intervals

The Myo-poller can gather data as fast as the Myo armband sends it. When working in Ubi-Interact, we want to define an interval for publishing the received Myo data. Our model needs 8 consecutive sEMG-data samples. The question arises whether there is a difference in classification, if we use different polling intervals for the training and test data. With shorter time steps between samples we can get more data in the same amount of time. This would save us recording time and help create bigger training sets.

We polled data from the same recording session in 3 different time intervals. One set with a sample every 100 milliseconds, one every 25 milliseconds and one as fast as we receive it from the Myo armband. We will call them train-100, train-25 and train-free. The recording happened simultaneously, so the set with the highest sampling rate is the smallest. Each set consists of the four training gestures described in the gesture set on page 21.

Polling interval	**Samples**	**Trainings steps**	**Accurcy**
100 ms	370	10,000	100%
25 ms	1480	10,000	99.39%
free	6810	20,000	98.49%

Table 6.1: Data from one trainings session was polled using different time intervals

A test with three sets of data with according polling intervals was performed. The results are displayed in table 6.2.

The highest accuracy can be found on the diagonal, where time-intervals of the training and test data match. The percentage varies around 4-7% between the test sets. This is not as much of a difference as we expected. Especially if we look at the difference of the sample size.

Training Data	**Test Data**		
	test-100	test-25	test-free
train-100	56.0%	52.0%	49.5%
train-25	59.0%	63.0%	61.2%
train-free	66.0%	62.5%	67.6%

Table 6.2: Same gesture data was polled in different time intervals trained afterwards

We can collect over 4 times more data using the free polling interval compared to the 25ms one in the same amount of time. This is a significant difference, especially for bigger sample sizes. Therefore, we will try to match the publishing interval we used in the Myo interface, but prioritize sample size first.

The following training sets are recorded with the free polling interval. In the final set, polling and publishing intervals will match.

6.3 How to perform gestures

The gestures vary in detection quality. While rock and the default gesture are classified well, scissors and paper have a low detection probability. We investigate whether this is based on how we perform the gestures. For example, does clenching the fist harder or stretching the fingers while doing the paper gesture influence the classification accuracy? We recorded two test sets. In the first set, the gestures were performed tense and in the other set more relaxed. Both sets have 10,000 samples and are trained for 30,000 steps. After that, a mixed set is created that consists of 5,000 gestures of each of the previous sets.

The trained models are tested with 2,000 relaxed and tense gestures.

Trainings-set	**test-relaxed**	**test-tense**	**test-combined**
train-relaxed	69.8%	74.6%	72.2%
train-tense	72.5%	76.9%	74.7%
train-mixed	78.1%	77.6%	77.9%

Table 6.3: Classification accuracy with differently performed gestures

The goal is to find a strategy to improve the classification of the paper and scissors gestures. Therefore, the accuracy of the individual gestures has to be considered. Overall, the mixed set has the best performance. It is also the only set that reaches over 70% correct recognition for every single gesture, if they are performed tensely. Surprisingly, the relaxed set also reaches over 70% accuracy for all relaxed gestures except the default one. Since the game has a time frame for input, instead of differentiating whether or not a gesture is performed, the low accuracy of the default gesture is irrelevant. So the best options are the relaxed and mixed training sets.

The mixed set has a higher accuracy on average, but for relaxed gestures, the mixed-set lacks in recognizing scissors, compared to the relaxed set. Since it feels more natural to do relaxed gestures and it is less tiring to record them for a big data set, we recorded a mixed set with mainly relaxed gestures for the final set.

Trainings-set	Gesture	test-relaxed	test-tense	test-combined
train-relaxed	default	44.5%	74.1%	59.3%
	scissors	72.7%	69.5%	71.1%
	rock	95.3%	89.7%	92.5%
	paper	71.8%	68.1%	69.9%
train-tense	default	91.8%	76.1%	84.0%
	scissors	57.9%	79.0%	68.4%
	rock	81.6%	93.6%	87.6%
	paper	61.4%	63.4%	62.4%
train-mixed	default	86.8%	79.8%	83.3%
	scissors	65.9%	76.8%	71.4%
	rock	86.4%	82.1%	84.2%
	paper	76.2%	72.2%	74.2%

Table 6.4: Classification in detail

6.4 Final data set

We recorded a set of 20,000 8x8 sEMG-samples from one person in a 10 milliseconds polling interval. $\frac{3}{4}$ of them were performed relaxed, $\frac{1}{4}$ tense. The model was trained with this data for 50,000 iterations. The following two figures show the accuracy during training and how each gesture from a test set was classified with this model.

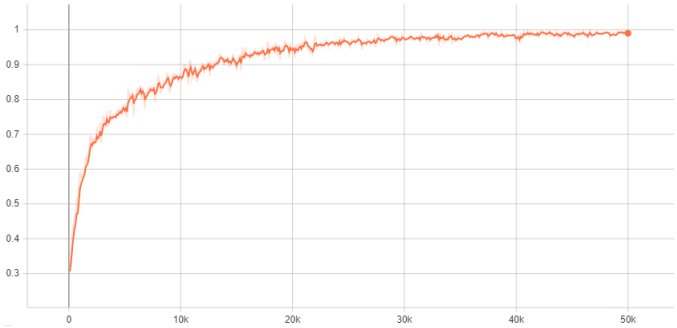

Figure 6.2: Trainings accuracy

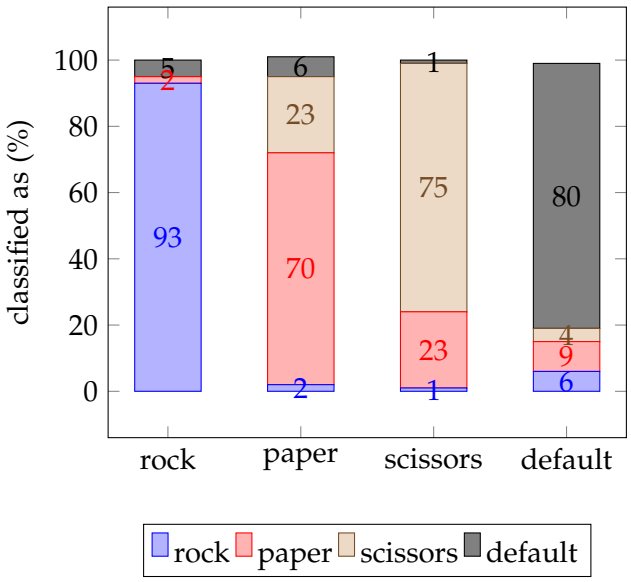

Figure 6.3: Accuracy per gesture

The rock gesture has the best classification rate with an accuracy of 93%. Paper and scissors are both correctly classified at least 70% of the time, but still have a 23% chance of being misclassified as the other gesture.

6.5 Generalization

We tested the gesture classification for people different from the person who recorded the training data. We collected sEMG data sets from two users and ran tests on the final model and some of the previous ones. Unfortunately, the highest accuracy we could reach was 49.5%. The model has overall weak generalization abilities. We also noticed that the classification accuracy is lower when we reposition the armband between recording the training and test set.

6.6 Gesture recognition in application

After training and importing the network in our application, we tested how well the gesture recognition works while playing the game. Each of the Rock-Paper-Scissor gestures was performed 50 times and noted how it was classified.

As we can see in figure 6.4, rock and paper are classified more than 50% of the time correctly. The classification accuracy for rock is 60% and other gestures are rarely misclassified (less than 5%) as it. Surprisingly, each gesture has a high chance to be classified as paper. It has the highest classification accuracy, while scissors has the lowest. Furthermore, scissors has almost the same chance of being misclassified as paper as it

has to be correctly classified. In total, we achieve a classification accuracy of 60% in the game. This is lower than the results of the Final data set section.

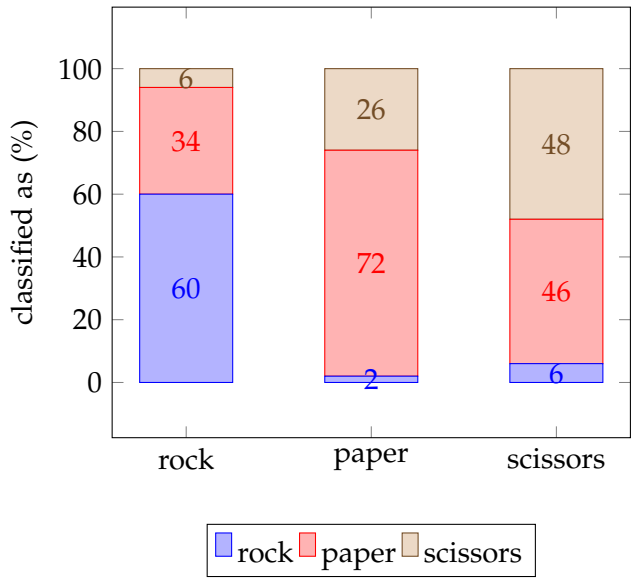

Figure 6.4: 50 gestures were classifed in application

To improve classification on the application side we tried, instead of resetting the sEMG-buffer after classification, to use a circular buffer approach for caching the data. We further tried to filter out input arrays where most of the gestures were classified as default and display a 'no gesture detected' message in that case. Unfortunately, none of these approaches had a significant influence on the classification. In all our tests rock was classified 55-60%, paper 60-70%, and paper 40-45% of the time correctly.

7 Conclusion

We achieved our goal of making an application to play Rock-Paper-Scissors using hand gesture input. We build an interface collecting sEMG data from the Myo armband. Furthermore, we converted, trained and loaded a Tensorflow model for sEMG classification and build the final game in Ubi-Interact. Unfortunately, only two of the gestures can be classified well in the application. The scissors gesture cannot reach a classification accuracy over 50% in the game.

Another issue is the practical use of the application, as the Myo armband is discontinued by now and the sEMG classifier model lacks generalization abilities. For a playable game, the model has to be trained with sEMG data from the individual player, which is impractical.

Other than that, the application works and can be used as a base for further work, which will be discussed next.

8 Future work

8.1 sEMG classification

8.1.1 Model

The gesture classification could be improved by using a different model. Unfortunately, the model conversion took more time then estimated in the beginning. So the model decision fell on the one that could be converted in a reasonable amount of time. Therefore, converting and comparing the models from Goll [Gol18] and Fromm [Fro18] could improve the classification.

8.1.2 Generalization

Another topic is the generalization issue. How can we make the model fit the current user the best? We could look at this problem from the Tensorflow side by training the model with a better dataset or try combining sets from different users. Furthermore, as we can train models in Tensorflow.js, we could do pre-training to fine-tune the model for each individual user. In general, doing some kind of pre-calibration could improve gesture classification.

8.2 Myo interface

We already have a Myo interface, so we could use it for other purposes. From using the IMU data for navigating the mouse cursor and using gestures for button clicks, to entirely different applications. The interface could be also integrated into Ubi-Interact's Virtual Reality modules.

8.2.1 Myo-Poller

The Myo-poller could be integrated into Ubi-Interact, so that all the Myo-related tasks could happen in one place. That data could be again exported as a CSV-file or being used in pre-training the model, for example.

8.3 Rock-Paper-Scissors Game

8.3.1 Using different input

We could operate the Rock-Paper-Scissors application with other input. For example, using the camera interface and image recognition techniques, similar to how it is done with the sEMG data now. Alternatively, a new interface could be implemented for other devices like the Leap Motion sensor.

8.3.2 Two Player

Another option would be to add a second player to make the game more interesting. Two Myo armbands or other input methods could be used to compete against each other.

List of Code Listings

List of Figures

List of Tables

Bibliography

[19] *Ubi-Interact*. 2019. URL: https://wiki.tum.de/pages/viewpage.action?spaceKey=infar&title=Ubi-Interact (visited on 09/09/2019).

[Aba+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. "TensorFlow: A System for Large-Scale Machine Learning." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1.

[BeB19] BeBobSensors. *Forte Data Gloves*. 2019. URL: https://bebopsensors.com/ (visited on 09/11/2019).

[Chu+17] L. Chuanjiang, R. Jian, C. Lulu, Z. Yanfei, Z. Chongming, and W. Peng. "Prosthetic hand control system based on MYO armband and control method of prosthetic hand control system." CN106890038 (A). 2017.

[Fro18] C. Fromm. "Hand-Gesture Classication via Neural Networks: Improving Accuracy with Continuous Use." Bachelor's Thesis. Technische Universität München, 2018.

[Fu12] K.-S. Fu. "Sequential Methods in Pattern Recognition and Machine Learning." In: 2012.

[Gol18] L. Goll. "An Adaptive Interface for Individual Surface EMG Classication Using an Articial Neural Network." Bachelor's Thesis. Technische Universität München, 2018.

[HAI19] H. F. Hassan, S. J. Abou-Loukh, and I. K. Ibraheem. "Teleoperated robotic arm movement using electromyography signal with wearable Myo armband." In: *Journal of King Saud University - Engineering Sciences* (2019). ISSN: 1018-3639. DOI: https://doi.org/10.1016/j.jksues.2019.05.001.

[Hal18] J. Hale. *Deep Learning Framework Power Scores 2018*. 2018. URL: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a (visited on 08/25/2019).

[Hig15] S. Higginbotham. *How Facebook is teaching computers to see*. 2015. URL: https://fortune.com/2015/06/15/facebook-ai-moments/ (visited on 09/09/2019).

[JM15] M. I. Jordan and T. M. Mitchell. "Machine learning: Trends, perspectives, and prospects." In: *Science* 349.6245 (2015), pp. 255–260. ISSN: 0036-8075. DOI: 10.1126/science.aaa8415. eprint: https://science.sciencemag.org/content/349/6245/255.full.pdf.

[Joh+17] V. John, M. Umetsu, A. Boyali, S. Mita, M. Imanishi, N. Sanma, and S. Shibata. "Real-time hand posture and gesture-based touchless automotive user interface using deep learning." In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. June 2017, pp. 869–874. DOI: 10.1109/IVS.2017.7995825.

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

[Lab18] T. Labs. *Myo armband*. 2018. URL: https://support.getmyo.com/ (visited on 08/25/2019).

[Mil05] K. R. Mills. "The basics of electromyography." In: *Journal of Neurology, Neurosurgery & Psychiatry* 76.suppl 2 (2005), pp. ii32–ii35. ISSN: 0022-3050. DOI: 10.1136/jnnp.2005.069211. eprint: https://jnnp.bmj.com/content/76/suppl_2/ii32.full.pdf.

[Mit16] D. Mitchell. *Surface Electromyography : Fundamentals, Computational Techniques and Clinical Applications.* Physical Medicine and Rehabilitation. Nova Science Publishers, Inc, 2016. ISBN: 9781536102024.

[Myt18] S. Mytrovtsiy. *Myo Armband Neural Network*. 2018. URL: https://github.com/exelban/myo-armband-nn (visited on 09/03/2019).

[Nas+16] E. Nasr-Esfahani, N. Karimi, S. M. R. Soroushmehr, M. H. Jafari, M. A. Khorsandi, S. Samavi, and K. Najarian. "Hand Gesture Recognition for Contactless Device Control in Operating Rooms." In: *CoRR* abs/1611.04138 (2016). arXiv: 1611.04138.

[NHJ15] K. Nymoen, M. R. Haugen, and A. R. Jensenius. "MuMYO - Evaluating and Exploring the MYO Armband for Musical Interaction." In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. NIME 2015. Baton Rouge, Louisiana, USA: The School of Music, the Center for Computation, and Technology (CCT), Louisiana State University, 2015, pp. 215–218. ISBN: 978-0-692-49547-6.

[Nie15] M. A. Nielsen. "Neural Networks and Deep Learning." In: Determination Press, 2015.

[Ove19] S. Overflow. *Developer Survey Results 2019*. 2019. URL: https://insights.stackoverflow.com/survey/2019#technology (visited on 08/25/2019).

[PJ15] D. Pyle and C. S. Jose. *An executive's guide to machine learning*. 2015. URL: https://www.mckinsey.com/industries/high-tech/our-insights/an-executives-guide-to-machine-learning (visited on 09/09/2019).

[PK] D. Park and H. Kim. *Muscleman: Wireless input device for a fighting action game based on the EMG signal and acceleration of the human forearm (2001).*

[RA11] S. S. Rautaray and A. Agrawal. "Interaction with virtual game through hand gesture recognition." In: *2011 International Conference on Multimedia, Signal Processing and Communication Technologies*. Dec. 2011, pp. 244–247. DOI: 10.1109/MSPCT.2011.6150485.

[RHM06] M. B. I. Reaz, M. S. Hussain, and F. Mohd-Yasin. "Techniques of EMG signal analysis: detection, processing, classification and applications." In: *Biological Procedures Online* 8.1 (Dec. 2006), pp. 11–35. ISSN: 1480-9222. DOI: 10.1251/bpo115.

[Sam18] Samsung. *How to use Palm Gesture to take Selfie on Samsung Mobile Device?* 2018. URL: https://www.samsung.com/sg/support/mobile-devices/how-to-use-palm-gesture-to-take-selfie-on-samsung-mobile-device/ (visited on 08/21/2019).

[Sch16] M. Schrimpf. "Should I use TensorFlow." In: *arXiv e-prints*, arXiv:1611.08903 (Nov. 2016), arXiv:1611.08903. arXiv: 1611.08903 [cs.LG].

[Sil+17a] I. N. da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves. "Artificial Neural Network Architectures and Training Processes." In: *Artificial Neural Networks : A Practical Course*. Cham: Springer International Publishing, 2017, pp. 21–28. ISBN: 978-3-319-43162-8. DOI: 10.1007/978-3-319-43162-8_2.

[Sil+17b] D. Silver, T. Hubert, J. Schrittwieser, and D. Hassabis. *AlphaZero: Shedding new light on chess, shogi, and Go*. 2017. URL: https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go (visited on 09/09/2019).

[SM15] M. Sathiyanarayanan and T. Mulling. "Map Navigation Using Hand Gesture Recognition: A Case Study Using MYO Connector on Apple Maps." In: *Procedia Computer Science* 58 (2015). Second International Symposium on Computer Vision and the Internet (VisionNet'15), pp. 50–57. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.08.008.

[Smi+19] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel, S. Bileschi, M. Terry, C. Nicholson, S. N. Gupta, S. Sirajuddin, D. Sculley, R. Monga, G. Corrado, F. B. Viégas, and M. Wattenberg. "TensorFlow.js: Machine Learning for the Web and Beyond." In: *arXiv e-prints*, arXiv:1901.05350 (Jan. 2019), arXiv:1901.05350. arXiv: 1901.05350 [cs.LG].

[Son19] C. Soni. *Role of Machine Learning in redefining Retail Banking*. 2019. URL: https://towardsdatascience.com/role-of-machine-learning-in-redefining-retail-banking-17796d350d9e (visited on 09/09/2019).

[Tec18] M. Techlabs. *8 Best Deep Learning Frameworks for Data Science enthusiasts*. 2018. URL: https://medium.com/the-mission/8-best-deep-learning-frameworks-for-data-science-enthusiasts-d72714157761 (visited on 08/26/2019).

[Tena] TensorFlow. *Importing a TensorFlow GraphDef based Models into TensorFlow.js*. URL: https://www.tensorflow.org/js/tutorials/conversion/import_saved_model (visited on 08/26/2019).

[Tenb] Tensorflow. *Model conversion*. URL: https://www.tensorflow.org/js/guide/conversion (visited on 08/30/2019).

[Tenc] Tensorflow. *Save and Restore*. URL: https://www.tensorflow.org/guide/saved_model (visited on 08/30/2019).

[Zha+11] X. Zhang, X. Chen, Y. Li, V. Lantz, K. Wang, and J. Yang. "A Framework for Hand Gesture Recognition Based on Accelerometer and EMG Sensors." In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 41.6 (Nov. 2011), pp. 1064–1076. ISSN: 1083-4427. DOI: 10.1109/TSMCA.2011.2116004.

[ZM17] Y. Zhang and O. Meruvia-Pastor. "Operating Virtual Panels with Hand Gestures in Immersive VR Games." In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Ed. by L. T. De Paolis, P. Bourdot, and A. Mongelli. Cham: Springer International Publishing, 2017, pp. 299–308. ISBN: 978-3-319-60922-5.