SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Action Recognition in Virtual Reality

Michael Vynogradov

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Action Recognition in Virtual Reality

Handlungserkennung in Virtual Reality

Author:	Michael Vynogradov
Supervisor:	Prof.Dr. Gudrun Klinker
Advisor:	Dr. Sandro Weber
Submission Date:	15.04.24

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 15.04.24 Michael Vynogradov

Acknowledgments

First and foremost I want to thank my advisor Dr. Sandro Weber for his continuous help throughout the writing of this thesis. Thank you for always answering my questions and reading my drafts.

I also want to thank Prof. Dr. Gudrun Klinker for supervising this topic and for her wonderful courses that I visited during my Bachelor studies.

I also want to thank my parents for their constant support throughout my studies and for making me the curious and hard-working person I am today.

I want to thank my girlfriend Hannah for her unwavering support. Your company and constant support were always an enormous help. Thank you for always making sure I was doing well.

Lastly, I want to thank all of my friends. Without your kind words and reassurance, this thesis would not be what it is today. Thank you for giving me a welcome distraction from the stress.

Abstract

This paper aims to solve the problem of recognizing user-performed actions in a Virtual Reality environment. Using the Unity game engine, a dataset of six actions was recorded by human beings. These six actions consisted of three movements, each performed with the left and right hands. The actions were chosen to be throwing, waving, and pointing. The dataset was constructed using the OpenVR Unity plugin API, which provided the position, velocity, and rotation of the VR headset and the handheld controllers. These were saved in a CSV and fed into a Neural Network with an LSTM (Long-Short-Term-Memory) Layer and three Dense Layers. After training the Neural Network multiple times with varying parameters, such as a higher Node count in the layers, adding Dropout layers, and a more extensive dataset, the Network was exported in an ONNX format and imported back into Unity, where the Barracuda framework was used to utilize the Neural Network. After importing the Neural Network, it can be fed live inputs. It then predicts the performed action and prints the result to the console. This paper was used as a proof of concept to show that action recognition in VR is possible. Possible topics that could be researched further include instructing a robot to perform actions based on the Neural Network's output or extending the Neural Network to support and recognize more actions.

Contents

1 Introduction

Virtual Reality (VR) has already found its place in various industries. It is used in design processes, data visualization, and the research of how specific actions impact a human being [2]. At the same time, using robots to perform dangerous or difficult tasks is also becoming more frequent, for example, in the aerospace industry [3]. Since both technologies have found their way into professional applications, combining them is an obvious next step, as maintaining more natural control over performed actions is crucial when working in extreme conditions. This paper aims to lay the foundation for fusing these two methods by creating a machine-learning model that can discern between different actions performed in a Virtual Reality space while only using data provided by the VR Headset and its controllers.

This paper created a dataset using the position, rotation, and velocity of the two handset controllers and the Head Mounted Display. To capture these data points, the game engine Unity was used. A custom script was written to capture this data. This dataset was then used to train a Neural Network, which uses the provided data to predict the action being performed. Through multiple tweaks and changes in the shape of the network, as well as the size of the dataset, the model achieved a peak accuracy of about 97 percent. This peak model was then exported back into Unity, where, through augmentation of the script, the performed action was saved in a file and sent to the Neural Network. The network then predicts the performed action and prints it to the console. With a 97 percent accuracy, on the self-recorded dataset, it achieves a similar accuracy to comparable papers that used videos as their dataset input [28].

1.1 Virtual Reality

Due to the high customizability of the virtual reality space, it lends itself to being modeled not just for entertainment purposes but also for a more serious context. Other papers, for example, have already explored the construction of a "digital twin" that is supposed to mimic the workings of a factory [10].

Hand movement and information are tracked from the two handheld controllers. Since the two controllers follow the hands, more natural hand movements like grabbing, shaking, and throwing can be realized just through the basic features implemented in every VR headset. More advanced headsets, like the Valve Index [9], can even detect which fingers are currently placed on the handheld controllers. Using VR for natural movement detection seems obvious. Since this thesis aimed to establish a proof of concept for action recognition, which could later be used to instruct robots, VR was chosen as the basis due to its benefits.

1.2 Motivation

With the release of more and more commercial VR headsets, VR has become not just a toy to play with in the gaming world but also a tool used in multiple professional applications each day [2]. Although much research has been done in action recognition using video-based techniques (see Related Work section), more research needs to be done on using just a VR headset to recognize performed actions. This thesis aims to provide a first approach to doing action recognition entirely in VR without the aid of other devices, like cameras or sensors. It will provide a base Unity implementation, which can later be extended to support more actions and be used in combination with the field of robotics. The implementation will be able to record actions and save them in a machine-readable format to easily facilitate the construction of a dataset. The implementation will also be able to read the recorded actions or get live action input and give this input to the Neural Network, which will then interpret and predict the performed action.

2 Related Work

Since the topic of using only the information provided by a VR headset and its handheld controllers for action recognition is poorly explored, this section will focus on papers that aim to achieve action recognition through video and similar means. Although these papers use different ways to capture their data, they were used as inspiration for constructing the dataset and the network for this thesis.

2.1 Neural Networks

The basis of this thesis, as well as other papers, is a machine learning model called a Neural Network. Neural Networks are a machine learning model with typical applications like Information Processing or Pattern Recognition [34]. A neural network consists of small computational units called nodes. Multiple of these nodes together form a layer. We can connect the nodes in each layer with the nodes from the following layer using weights. These weights get adjusted when the neural network is being trained. These adjustments happen through forward propagation, followed by backward propagation. Forward propagation uses the weights in the current neural network to calculate a possible result, a prediction. The prediction is then verified, i.e., does the predicted action's label match the labeling for the recorded action? The backward propagation step is executed once the forward propagation has been performed. In this step, the feedback on whether the prediction was correct is used to adjust the weights, tweaking them to increase the likelihood of an accurate prediction. Forward and backward propagation is performed over multiple elements from a prerecorded dataset [14].

2.2 Action Recognition using Video-Based Techniques

Many papers provide research into the topic of action recognition using video. This section will focus on a few documents that provide the most crucial input for this thesis.

2.2.1 Action Recognition using LSTM

The paper by Liang et al. [22] makes a point about using a so-called Long-Short-Term-Memory (LSTM) layer to perform video action recognition. This paper was the basis for deciding to use an LSTM layer in this thesis since it has substantial benefits in terms of spatial and temporal context. Since the paper also focused on predictions on video, they also used a Convolutional Neural Network; this does not apply to this thesis since no video camera was used. Due to how the LSTM layer works [17], it is ideally suited to be used in action recognition. The LSTM layer uses three gates: an input gate, a forget gate, and an output gate. This structure can work exceptionally well on a time series of events since it can decide which information to keep and discard from the presented data.

2.2.2 Action Recognition in a VR Space

The first influential paper was the paper by Dallel et al. This paper focused on the practice of constructing a so-called "digital twin" for industrial applications. In this paper, the researchers used the VR environment for data collection instead of action recognition. The action recognition was then performed using RGB cameras and sensors. The VR environment was only used to allow for higher immersion and more accessible performing of the action by the participants. The actual data was captured through RGB video and sensors, which were attached to the participants. This approach inspired using the handheld controllers' velocity as a basis for the dataset, which will be elaborated on in the implementation part of this thesis.[10]

The second exciting paper was the paper by Li et al. This paper focused on action recognition using RGB video and positional data provided by a VR headset's Head-mounted display (HMD). Since it mostly used RGB video for prediction, this paper only inspired the idea of using the position provided by both handheld controllers and the HMD.[21]

2.2.3 Other Methods for Action Recognition

The research on performing action recognition can go in many directions. Some follow the idea of using video-based techniques but use various types of Neural Networks. For example, Lu et al. [27] proposed a method of using an LSTM but augmenting it with a so-called YOLOv4 network, which is also often used in the video-based action recognition field, to the great success of a 97.87% accuracy. Another standard method is using a Convolutional Neural Network (CNN) when working with video.

This approach divides the video into frames, which are then condensed for more straightforward computation by the Neural Network. The paper by Wang et al. [33] focuses on using a bigger CNN that pools together multiple frames of a video and predicts the action based on that frame pool. Another more advanced approach was chosen by Ji et al. [19]. Their approach used a so-called 3D CNN. Similar to an LSTM, the 3D CNN incorporates spatial and temporal components into the video-based prediction process, leading to about 90.2% accuracy. A different direction was pursued by Hirota et al. [16]. They chose to use the video feed from a camera attached to a VR headset and the position of an object in the VR space to predict if the user is grasping the object. Their results showed that adding an object into the virtual space and feeding that information into the Neural Network also helped increase the accuracy.

Another common approach is the construction of a skeleton based on the video data provided. With this approach, the skeleton is fed into a neural network, and the prediction happens on the skeleton data instead of on the video. The paper by Elias et al. [13] focuses on constructing a 2D skeleton and using an LSTM to predict actions using the PKU-MMD Dataset, which will be explained in the following chapters. The paper also aimed to get results comparable to those obtained by using a 3D skeleton, which it achieved with an accuracy of 97.4%. Ye et al. [36] chose the 3D skeleton approach for their paper. They constructed a 3D skeleton from video input and then used an LSTM network to predict the performed actions, achieving a 99.3% accuracy on their chosen datasets. Yan et al. [35] chose another 2D-based skeleton approach but used a Spatial Temporal Graph Convolutional Network for the action recognition instead of an LSTM network. This decision led to them achieving a 71.7% accuracy on their chosen datasets. The skeleton-based approach was not chosen for this thesis due to the assumption that the VR hand- and headset could deliver comparable information with less effort.

Approaches to action recognition are not limited to video or VR-based technologies. The paper by Schröder et al. [31] utilized an AR headset instead of a VR headset to predict in which progress stage the construction of a birdhouse was and what actions the user is currently performing, to give recommendations on which action to perform next. They also used CNN to make a prediction. This thesis decided against an AR approach since the precision and accuracy of the handheld controllers were considered to be higher than what could be extracted from an AR headset. A different idea was pursued by Li et al. [20]. In their paper, they used sensors, like accelerometers, to extract feature data through the k-means clustering and then fed it into an XGBoost network to be predicted. This machine-learning approach is entirely different from the other papers discussed in this section since those papers all used neural networks. However, it provides a good insight into the fact that action recognition is a vast field that allows for many different possible ways to solve the action recognition problem.

2.3 Dataset

The following papers focus on the creation of different datasets for human movement. It will be outlined how those papers influenced the dataset creation for this thesis.

2.3.1 HARTH

The paper by Logacjov et al. focuses on creating a human activity recognition (HAR) dataset. The dataset consists of videos and accelerometer data of participants performing everyday actions. It also explored different machine learning models, including LSTM. The results showed that Support Vector Machines had the best results, but LSTM and the other models also achieved excellent results. This paper was influential by providing another instance of an LSTM model being used successfully and the idea of capturing things like velocity and position from the VR headset with its handheld controllers.[24]

2.3.2 PKU-MMD

This paper by Liu et al. created a multi-modal dataset of everyday actions. Among these modalities are RGB video, infrared radiation, and skeletal data. The approach to using skeletal data was used as a base idea for a possible dataset in this thesis. This idea was later scrapped for a computationally less expensive method of using the handheld controllers' velocity, rotation, and position instead of constructing a skeleton from the data provided. This dataset inspired the recorded actions for this thesis, containing upper body actions like "wave hand" and others.[23]

2.3.3 Padding

The last paper is by Dwarampudi et al. and focuses on the most optimal way to pad a dataset. LSTM Networks expect a certain length of input at all times since not all actions have the same length. Some of them must be padded so the LSTM can compute them. The paper focuses on which type of padding, front or back, is best for LSTM networks. The result of the study was that padding the sequence at the front yields the best results. Due to these findings, this thesis, as shown later in the implementation, also uses padding at the front of the sequence.[12]

3 Implementation

After discussing possible approaches to the project in the Related Work section, this section will focus on the devised plan for the dataset and the neural network. This section will also present the recorded results. The source code for the entire project can be found on Gitlab under the following link: https://gitlab.lrz.de/vyno/bachelor-thesis.

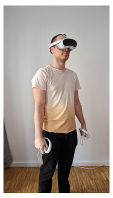
3.1 Tools

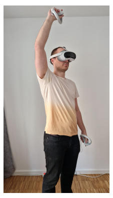
The Unity game engine was chosen to implement the project due to the availability of an easy-to-use VR-Software Development Kit (SDK), the engine being free for smaller projects like this thesis, and prior experience with the engine. The scripts were written in the C# programming language, and the integration of the VR Headset into the engine was handled by the Unity OpenVR plugin [7]. The default prefab and Input Manager for VR were used for any VR interactions in Unity, like the action recording. The used Unity Version was 2022.03.16f1.

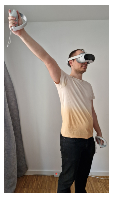
The Neural Network was constructed using Google's Tensorflow library [11] and trained in the Google Colab environment using a T4 GPU by Nvidia. The training happened on Google's cloud servers. Python was used to code the entire notebook and construct the neural network.

To run the Neural Network in Unity, Unity Barracuda was chosen [6]. The network was exported from Google Colab in the ONNX file format requested by Unity Barracuda and then imported into Unity, where it was then used in a C# file to predict given actions. Further reasoning as to why Unity Barracuda was chosen will be provided later in the Implementation chapter.

As for the hardware, the Pico 4 was chosen [26] due to it being readily available during this thesis and being very easily compatible with the other laptop hardware used to write the Unity implementation part. To connect the headset to the laptop, the Streaming Assistant [25] and SteamVR [8] were used.


(a) Start of action


(b) First midpoint of action


(c) Second midpoint of action

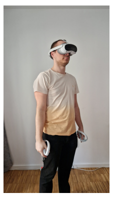
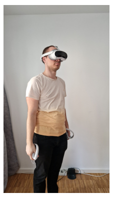
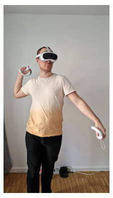
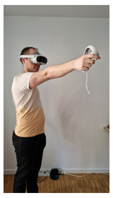

(d) End of action

Figure 3.1: Wave being performed


(a) Start of action


(b) First midpoint of action


(c) Second midpoint of action

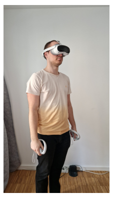
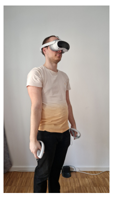

(d) End of action

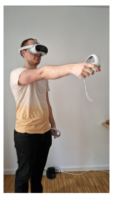
Figure 3.2: Throw being performed

3.2 Dataset

The first problem this thesis aims to answer is constructing a suitable Dataset. The questions about which actions to record and which features to extract were the first to be considered. The research started with finding papers that focused on constructing datasets and seeing what kind of datasets similar papers used. These papers have already been discussed in the Related Work chapter of this thesis. Through paper research, the possibilities for dataset construction were narrowed down to two possible options: constructing a skeleton through the use of inverse kinematics, similar to the papers discussed in section 2.2.3 of the Related Work, and using the position of the different joints for the predictions, or using the provided velocity and rotation by the handheld controllers similar to the sensor-based action recognition. The decision was made to use the information provided by the VR Headset and its handheld controllers since it was computationally less intensive to read the information than to construct a skeleton from the information presented.

The next problem that was tackled was which actions to record. Due to the headset only being able to accurately track and provide information for the upper body, the actions were chosen to be just upper-body actions. The three actions chosen were waving with


(a) Start of action


(b) First midpoint of action

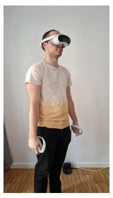

(c) End of action

Figure 3.3: Point being performed

the hand, throwing an object, and pointing in a direction. The actions are depicted in Figure 3.1, Figure 3.2, and Figure 3.3. As seen in the Figures, the actions always have the same start and stop pose. This decision was made so the Neural Network could not deduce the actions from differing start and stop poses but would have to analyze the entire action run. The throw and point actions were chosen due to their similarity. They have different trajectories and lengths, but the midpoint in Figure 3.2c and the midpoint in Figure 3.3b are very similar to each other, both in position and rotation of the handheld controllers. The wave action was chosen due to its similar length to the throw motion and because it is a prevalent upper-body action. These actions helped determine whether the Neural Network can discern between similar actions and cover some prevalent actions that can be used for interaction in the VR space. Furthermore, the choice was made to record each action on the right and left sides and treat these recordings as separate actions. This decision was made with the continuation of this work in mind since the individual prediction of the sides can be used to better discern what the user is trying to achieve. For example, if an object is placed on the left in VR space and the user points with the right hand, the application could discern that the user is not trying to point at the given object but something else. Due to this thesis's scope and given time frame, the size of the dataset was limited. The choice was made to perform each action fifty times on each side. This would lead to a sizeable dataset of 300 total performed actions per participant while still being able to be recorded in the given time frame.

After deciding what actions to record, the next step was determining which information to extract from the UnityVR framework. All actions were recorded in the UnityVR package's basic sample scene. The position, velocity, and rotation of the two handheld controllers and the head-mounted display (HMD) were extracted. This decision was based on research into other papers' different action recognition approaches. Due to their prevalence and performance, two methods were combined: the skeletal-based research inspired the use of the position, and the sensor-based research inspired the use of the velocity and rotation of the VR handheld controllers and the HMD. Another advantage of these parameters is that they are device agnostic - every modern VR headset provides this information, so in theory, the dataset creation and action prediction can be performed on any modern headset. All the parameters were read from the UnityVR SDK. A C# script in Unity was created to capture all parameters. The script records all three devices, two handheld controllers, and the HMD, with a 30 Hz sample rate. All the data is stored in lists while the recording is running. Once the recording is done, all lists are saved into a CSV file in a user-provided location. In the beginning, two participants were recorded for the training set. Each participant signed a consent form, allowing the use of their recorded data for this paper. One person from the original dataset recorded 20 extra actions as a test set. A velocity calculation script was also created due to a bug that set the velocity of the HMD to zero at all times. The script was written in Python and calculates the velocity simply from the current position, the position in the last time frame, and the passed time.

3.3 Neural Network

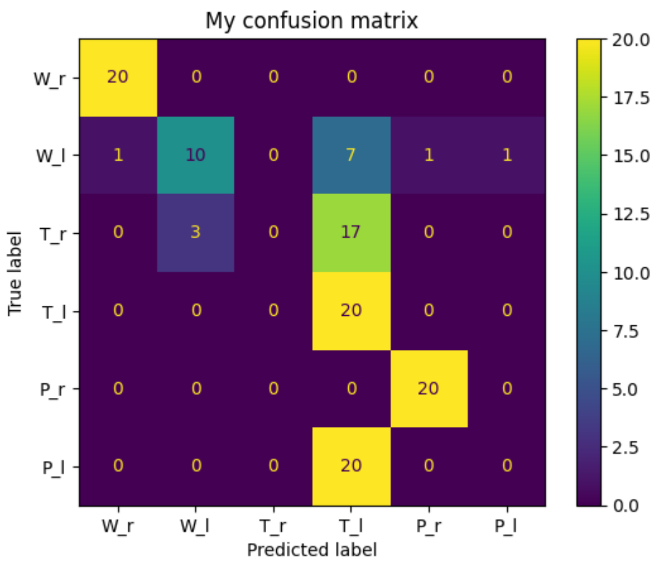
The following section will focus on the planning and execution of the Neural Network. It will also showcase the achieved results during testing.

3.3.1 Planning of the Network

After the dataset was finalized, the Neural Network was created. As the Related Work section explained, the Neural Network utilizes an LSTM layer to enable time series prediction. Two regular deep layers follow the LSTM layer and end in a prediction deep layer, producing the prediction. The LSTM layer and each of the deep layers consist of 100 nodes each and use a RELU activation function. Due to the dataset not being overly large, 100 nodes were deemed a large enough layer size to accurately predict the actions performed. The RELU activation function was chosen because it is a widespread and well-performing activation function [1]. For example, the paper by Lu et al.[27] also uses the RELU function for their predictions. The Network optimizer was selected as the ADAM optimizer because it is a very performant and broadly used optimizer [18]. The loss function was chosen as the categorical cross-entropy function. This decision was made on the basis that multiple different categories of actions are predicted, for which this loss function is typically used [29]. Due to the stochastic nature of the Network [4], the decision was made to rerun each dataset 10 times and determine the average accuracy and peak accuracy for that run. Since the amount of data points for individual actions differed, a preprocessing step was taken. Every dataset with a datapoint count of less than 41 is padded to exactly length 41 with a pre-padding of zeros. The number 41 was chosen, due to it being the amount of data points of the longest performed action in the dataset. The reason for pre-padding was already explained in the Related Work section of this thesis. This enabled the LSTM to predict executions of variable length.

3.3.2 Training and Results of the Neural Network

The training and test set were executed after constructing the Neural Network in TensorFlow. This led to an average accuracy of 56.58% and a peak accuracy of about 60.83%. The first improvement idea was removing the HMD data to see if the dataset was too large for the network. The result was an average accuracy of 56% and a peak accuracy of 62.6%. This attempt showed that the poor accuracy was not due to the dataset being too large for the network, but it had to be due to other reasons. After this, a confusion matrix was created based on the best-performing model, as shown in Figure 3.4. A confusion matrix shows which action was predicted and which was performed [15]. The x-axis is labeled with the predicted actions, while the y-axis is labeled with the actual actions. Predicted and actual labels have to be equal to achieve a true positive.



True label \ Predicted label	W_r	W_l	T_r	T_l	P_r	P_l
W_r	20	0	0	0	0	0
W_l	1	10	0	7	1	1
T_r	0	3	0	17	0	0
T_l	0	0	0	20	0	0
P_r	0	0	0	0	20	0
P_l	0	0	0	20	0	0

Figure 3.4: Basic Confusion Matrix

Due to the network's poor performance, the next step was to find ways to improve the network. The first attempt at improving the network was using so-called Dropout layers. These layers are commonly used to avoid overfitting the dataset. In this practice, the Neural Network sees the training dataset so many times that it gets very good at recognizing only this dataset and no others. Dropout layers "randomly drop units from the network during training" [32], which then helps avoid overfitting. A Dropout layer with a 50% dropout rate was put between the LSTM layer and the following deep layers, as well as between every deep layer. Also, another deep layer was added to the network to raise accuracy by growing the network size and allowing more data to be computed. The average accuracy only marginally improved, with the new average being 56.5%, while the peak accuracy was raised to about 65.83%. Figure 3.5 shows the confusion matrix for the best model.

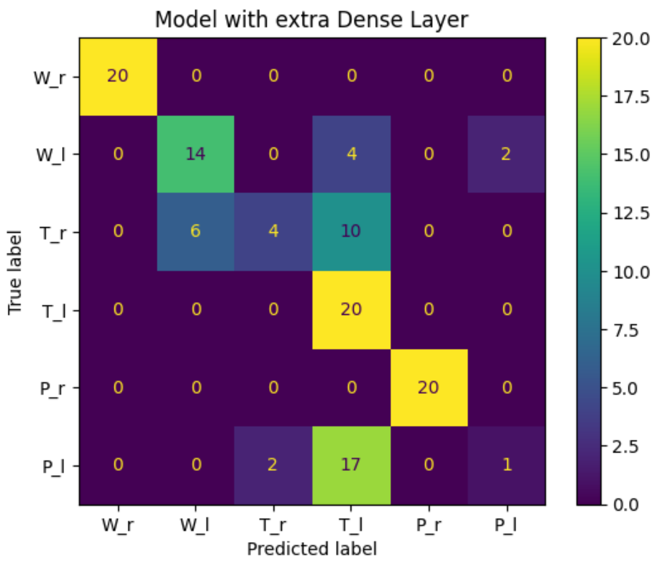


Figure 3.5: Confusion Matrix for added Dropout layers and extra Dense Layer

Another attempt to improve accuracy was made by increasing the size of the layers to 200 nodes each since the prevalent suspicion was that the network could not handle the amount of data passed through it. An increased node amount could allow more data to be handled, bringing higher accuracy. The increase from 100 to 200 nodes in each layer brought a rise to 57.33% average accuracy and a peak accuracy of 64.16%. Although the peak accuracy was slightly lower, the average accuracy increased somewhat. As shown in Figure 3.6, the predicted actions were close to the previous Figure 3.5 but still showed a slight improvement in certain actions.

Since none of the previous tweaks to the network brought a significant enough increase in the accuracy to be considered successful, other methods had to be explored. As mentioned before, the dataset size was assumed to be too small, which led to the addition of dropout layers. Two more participants were added to the training dataset to remedy the problem of a too small dataset. This resulted in four people in the dataset and increased the average accuracy to 68.75% and the peak accuracy to 72.5%. This was a tremendous success and can also be seen in the confusion matrix in Figure 3.7. Due to time constraints for this thesis, one of the improvement approaches had to be selected to be pursued further. Due to the significant increase, recording more participants for the dataset was chosen as the best solution to further increase the accuracy.

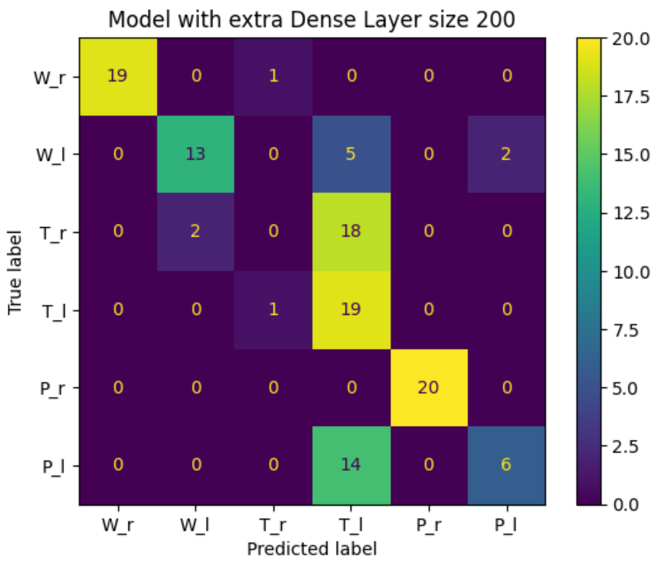


Figure 3.6: Confusion Matrix with 200 Nodes in each layer

Due to the reasons outlined previously, three more people were added to the dataset. The added participants increased the average accuracy to 78.08% and the peak accuracy to 81.67%. As shown in Figure 3.8, almost all actions are correctly recognized, except for the right throw, which is still predicted as a left throw. Since this error could be observed in all of the tests done so far, this led to the assumption that the test set was erroneous.

As described before, the test dataset was recorded by one of the participants already in the dataset. Ideally, someone who did not record actions for the training set would record the test set. The cross-validation approach was chosen to better reflect its performance on a user not in the dataset and remedy the error in the previous test dataset. The cross-validation chosen here is the so-called "leave-one-subject-out" cross-validation, also used in the HARTH paper by Logacjov et al. [24]. With this approach, one of the subjects is taken out of the dataset to be used as a test set, while all other participants combined are used as the training set. The results differed between which participant was taken out of the network, and the results will be shown and discussed in the Analysis section.

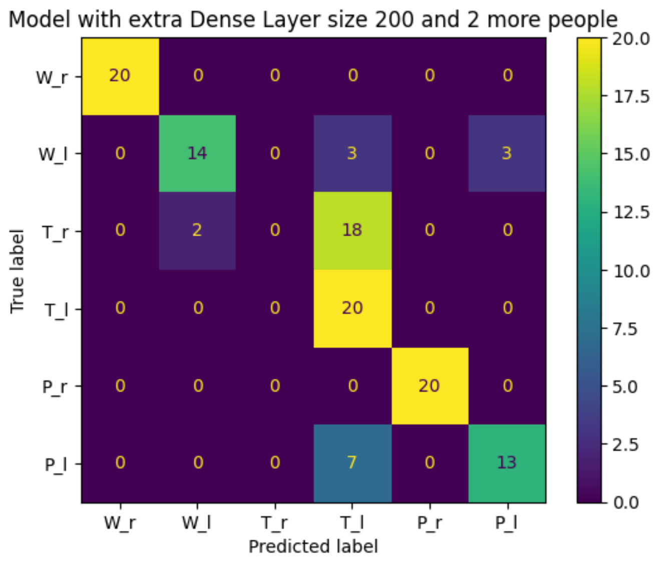


Figure 3.7: Confusion Matrix for 4 people in trainingset

3.4 Unity Implementation

Once the Neural Network was trained, the next step for this project was to tie the network back into Unity. Multiple approaches were considered. The first approach was to use the Google Vertex AI platform [5]. This approach would have entailed hosting the network in a Google-provided cloud and communicating with it through REST API calls. This solution was not chosen due to the problem of correctly sending the recorded data to the server and the assumption that the network could be executed efficiently on local hardware. For the execution on local hardware, a laptop with the following hardware was used: an AMD Ryzen 5 4600HS CPU, 16GB of RAM, and a Nvidia GTX 1650Ti as the graphics card. Unity Barracuda was used to incorporate Neural Networks into Unity. Barracuda was chosen due to its easy integration of self-trained networks into Unity and the advantage of running the Neural Network locally. This leads to less reliance on having a stable internet connection for the predictions. It is beneficial

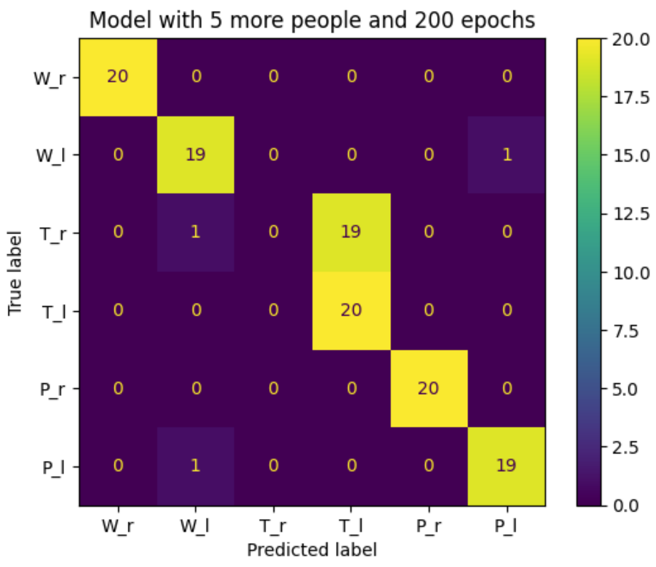


Figure 3.8: Confusion Matrix for 7 people in trainingset

for future applications of this method, for which a stable internet connection cannot always be guaranteed.

The first step to being able to use the Neural Network in Unity was to export it into the ONNX format. To export it from the Google Colab environment, the tfonnx tool was used [30]. Once the model was exported, it was added to the Unity project as a game object. The game object was then tied into a Unity script, which takes in lists of actions performed and feeds them into the network. The network then gives out a prediction, which is displayed on the console. Functionality was added to read the performed actions from a prerecorded CSV file. In this use case, the CSV file is transformed into lists of the actions performed and then presented to the network. A live prediction functionality was implemented as well. For the live prediction functionality, the Unity script records all the actions performed by the user and then saves them into a CSV for later inspection. After the save, the action is sent to the Neural Network, which predicts the performed action and displays it in the console. The time between the performing of the action and the prediction being displayed is very fast, roughly 181 ms on average.

4 Analysis

This chapter of the thesis will analyze the previously reported results, show the cross-validation results, and examine them. It will also explain why certain decisions were made in the Implementation section.

4.1 Smaller Tweaks to the Neural Network

As was outlined at the beginning of Chapter 3.3.2, the first attempt at the neural network delivered average results of a peak accuracy of 60.83% and an average accuracy of 56.58%. As seen in Figure 3.4, the wave right, point right, and throw left actions were recognized perfectly, while wave left was only predicted correctly about 50% of the time. The most significant anomalies in this attempt were the throw-right action as well as the point-left action. The problem with the point left action being recognized as throw left was assumed to be due to the similar middle point of the actions, as seen in Figure 3.3b and Figure 3.2c. The problem with the throw-right action caused suspicion that something might not be correct in the test dataset. Still, it was initially dismissed as a possible difficulty of recognizing more complex movements, like a throw, while also correctly recognizing the side it was on.

Due to the prevalent assumption being that the dataset was too small and the actions were too complex for the network, the network was expanded by another layer, and dropout layers were added. These dropout layers were added to help combat overfitting issues on such a small dataset. The increase in peak accuracy, from 60.83% to 65.83%, was seen as an improvement in the right direction. As shown in Figure 3.5, the actions of wave right, throw left, and point right are still correctly recognized. The previously wrongly predicted actions saw slight improvements. Wave left rose to a 70% accuracy, from 50 %, throw right improved to 20%, from 0%, and point left gained 5% accuracy. This showed that although no significant improvements were made, the tweaks successfully increased the network's correctness.

Due to the fact that the right throw was still not recognized correctly, the decision was made to increase the overall size of the network from 100 nodes per layer to 200 nodes per layer. It was assumed that this change would help with recognizing the more complex throwing action and allow for better differentiation between left and right. As shown in Figure 3.6, this leads to different results for different actions. While wave left and wave right got one less prediction correct, point left increased to an accuracy of 30%, from 5%. Throw right decreased in accuracy, being predicted as throw left 50% of the time. Point right stayed entirely accurate. Apart from the increase in accuracy of the point left action, this change did not have a significant enough impact on the predictions to be considered a success. This change did show that the network was not too small to handle more complex operations. Instead, the assumption was made that the biggest problem was the dataset size.

4.2 Adding more People to the Dataset

With only two people in the training dataset, the assumption was made that more data was needed to successfully improve the accuracy. This assumption was based on other related papers. For example, the paper by Logacjov et al. [24] had twenty-two participants, and the paper by Liu et al. [23] had sixty-six participants. Also, simpler actions seemed to require more data to be predicted correctly all the time. Due to the time constraints for this thesis, the decision was made to focus on adding more people's recorded actions to the dataset. Two more people were added to the training dataset, as a test to see how the accuracy would change. This was the most considerable success so far. It increased the peak accuracy to 72.5%, from 64.16%, and the average accuracy to 68.75%, from 57.33%. As shown in Figure 3.7, point left had the most significant prediction improvement, rising to an accuracy of 65%, from 30%. Although the other actions did not see as significant an increase in accuracy as point left, the vast improvement in average accuracy showed that the problem of the dataset being too small was the most significant. At this point, the suspicion of an erroneous dataset was still considered due to throw right still being mostly recognized as throw left. However, due to the still poor performance of other actions, like wave left, this was still dismissed as possibly being due to the small size of the dataset. It was assumed the mistake with throw right would disappear, once the training dataset was expanded.

To further increase the size of the training dataset, three more people were recorded. This put the total number of participants at seven people. As seen in Figure 3.8, this increased the model's accuracy tremendously. The average accuracy increased to 78.08%, from 65%, and the peak accuracy was raised to 81.66%, from 72.5%. Almost all actions were recognized successfully, with one outlier for point left and wave left each. The only action still consistently being predicted wrong was throw right. This solidified the suspicion that there was a mistake in the test dataset concerning throw right. A cross-validation approach was chosen to remedy the error in the dataset and test how the network would perform when trained on a person who is not in the training dataset.

4.3 Cross Validation

As mentioned before, the chosen cross-validation approach was the "leave-one-subject-out" method. In this approach, one of the participants is taken out of the dataset to be used as the test set, while the others are used as the training set. In this thesis, every participant was used five times as the test set, and a new model was created every time. The confusion matrix of the best-performing model for each test participant can be seen from Figure 4.1 to Figure 4.7. The following sections will analyze general trends, which can be deduced from the confusion matrices.

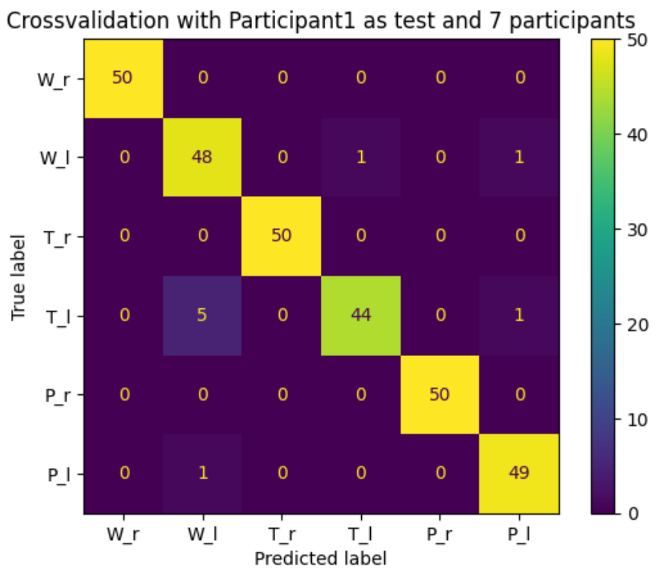
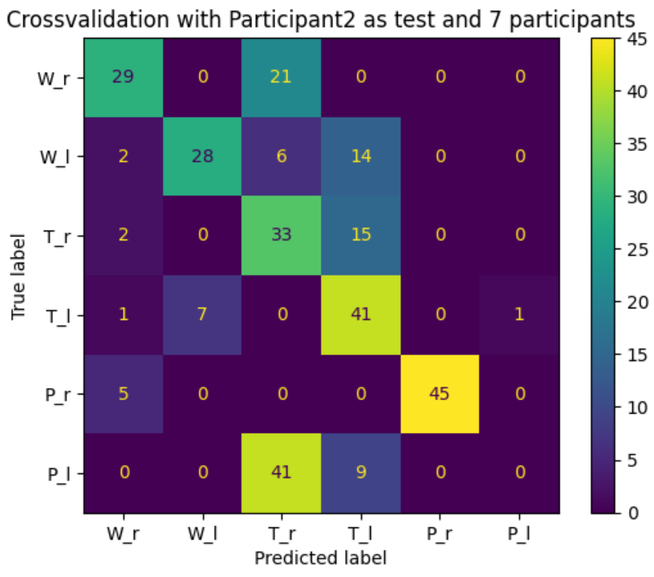


Figure 4.1: Cross Validation with Participant 1 as Testset

Participant 1 has the best results out of the dataset, as seen in Figure 4.1. The accuracy for the displayed model was 97%, and all actions, except for throw left, are perfectly recognized. Throw left is only incorrectly predicted in 6 out of the 50 test cases. This shows very promising results for the model. It is assumed that the accuracy of this participant is high because they performed the actions very similar to how they were instructed. Throw left might still be recognized wrongly in some cases due to the participant being only one of two lefthanded people that participated in this thesis. This could lead to a certain bias towards right-handed actions being predicted more accurately since they were more often performed with the participant's dominant hand.

Participant 2 achieved a peak accuracy of 58.66%, as displayed in Figure 4.2. This is the lowest peak accuracy out of all the participants. This participant was also the one who recorded the original test set. The wave right action and the throw right action have a significant overlap. It is assumed this is due to similar velocities during the mid-parts of both actions. A similar thing can be seen in the prediction of wave left as throw left, probably due to similar reasons. Throw right is often also recognized as throw left. As mentioned before, the participant also recorded the test dataset, which could lead to the assumption that a similar error to the test dataset is also present here. This participant is one of three who has mixups between the right and the left side actions. The fact that only two other participants expressed similar errors supports the theory of an error with this participant. The errors in the other users' predictions are assumed to be due to errors brought into the training set by this participant. The point left action was not correctly predicted at all. Instead, it was assumed to be either throw right or throw left. This could be because the middle parts of the actions, seen in Figure 3.2b and Figure 3.3b, are very similar. The error of predicting right instead of left is assumed to be similar to the errors above.



True \ Predicted	W_r	W_l	T_r	T_l	P_r	P_l
W_r	29	0	21	0	0	0
W_l	2	28	6	14	0	0
T_r	2	0	33	15	0	0
T_l	1	7	0	41	0	1
P_r	5	0	0	0	45	0
P_l	0	0	41	9	0	0

Figure 4.2: Cross Validation with Participant 2 as Testset

Participant 3 achieved a peak accuracy of 91%, as displayed in Figure 4.3. Most actions are correctly recognized, with outliers being wave right, wave left, and throw left. The two wave actions have a significant overlap with the pointing action. This could be due to the participant's improper execution of the action by waving, not as described in Figure 3.1, but rather holding the hand in front of them while waving. This would lead to the midpoints of wave and point being very similar in position. In papers also focused on dataset creation, like the paper by Logacjov et al. [24], participants were asked to perform everyday activities, which led to different kinds of execution. This paper had a much larger sample size of twenty-two people. This shows that with a bigger sample size, even the imperfect execution of the actions can be correctly recognized. The throw left action was also recognized 22% of the time as the throw right action, an error also expressed in the test set with Participant 2. This could be due to reasons similar to those previously described with Participant 2, where erroneous data might have been recorded.

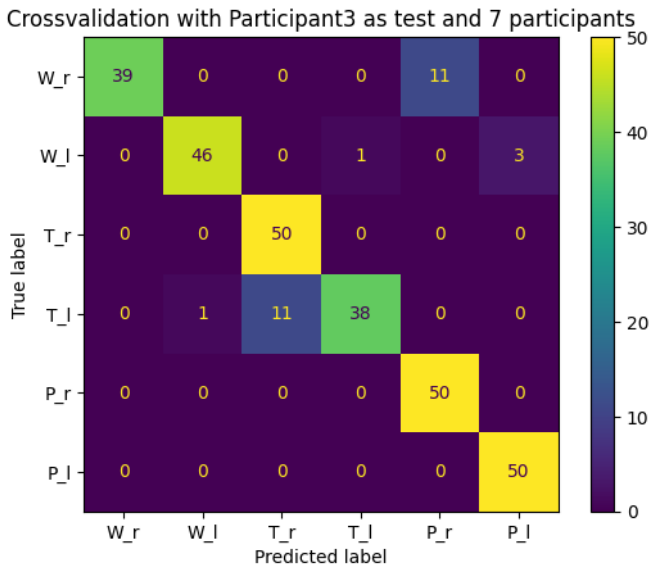


Figure 4.3: Cross Validation with Participant 3 as Testset

In Figure 4.4 it can be seen that Participant 4 achieved a peak accuracy of 64.64%. As can be seen in the Confusion Matrix, most actions are correctly predicted, apart from some outliers. The two wave actions are not predicted correctly. Both have a significant overlap with the pointing and throwing action on their respective side. This could be due to similar reasons outlined in the discussion on Participant 3. As discussed in the analysis of Participant 2, the wave action and the throw action could be detected as the same, due to similar velocities at the midpoints of each action. Some of the wave right actions are also predicted as throw left, which could be due to smaller training set errors caused by Participant 2.

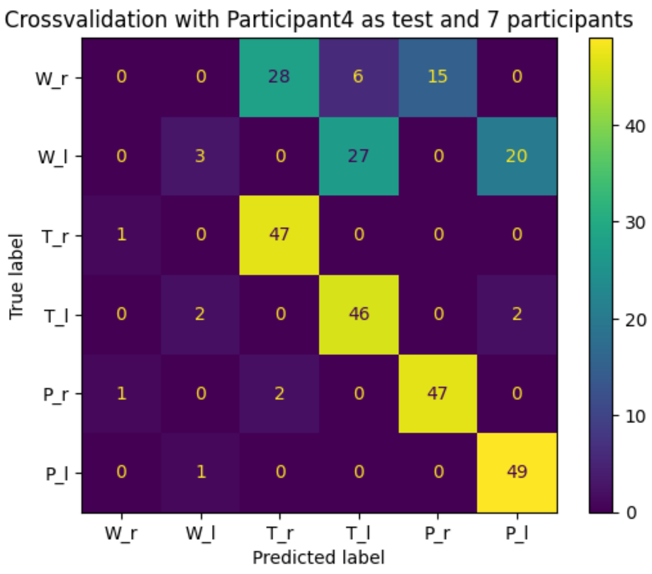


Figure 4.4: Cross Validation with Participant 4 as Testset

Figure 4.5 shows the results for Participant 5. The peak accuracy of this participant was 87%. The Confusion Matrix shows that most actions have been recognized correctly, with a few outliers. The actions with the lowest accuracy were the wave right action and the throw right action. Like with Participant 4 and Participant 3, the wave action is recognized as the point right action. As previously stated, this could be due to incorrect execution of the actions while recording. Also, due to the points outlined in the discussions for Participants 2 and 4, wave and throw actions are recognized similarly.

The results of Participant 6 can be seen in Figure 4.6. The participant achieved a peak accuracy of 62%, placing it as the second lowest peak accuracy. This participant's confusion matrix shows errors that are similar to the ones previously described. Both wave actions are sometimes recognized as the point action, which could be due to the way the actions were performed, similar to the case of Participant 3. Also, both wave actions are sometimes recognized as the throw action on the respective side. This can be due to the same problem outlined with Participant 2. The same problems also arise when looking at the throw and point actions. While recording, this participant did not fully follow the instructions. This can be seen in the results above. It also shows that

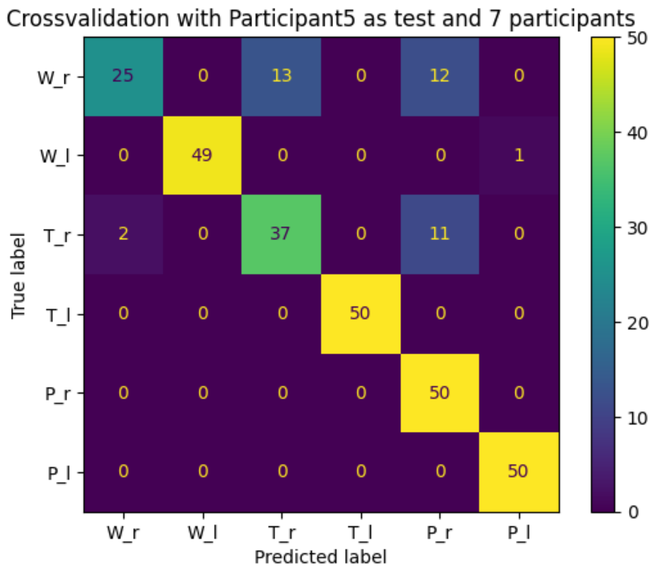


Figure 4.5: Cross Validation with Participant 5 as Testset

the network only manages to predict actions that are very close in performance to the instructed way of execution.

The last participant to be analyzed is Participant 7. Their peak accuracy was 84.33%, and the resulting confusion matrix can be seen in Figure 4.7. Apart from some outliers in throw right, throw left, and point right, the accuracy is very good. The action with the lowest accuracy is point left. As mentioned in the discussion for Participant 2, the overlap between point and throw could be due to the midpoint of the actions being very similar. The overlap with wave left could be due to the performance of the action when recording, similar to Participant 3.

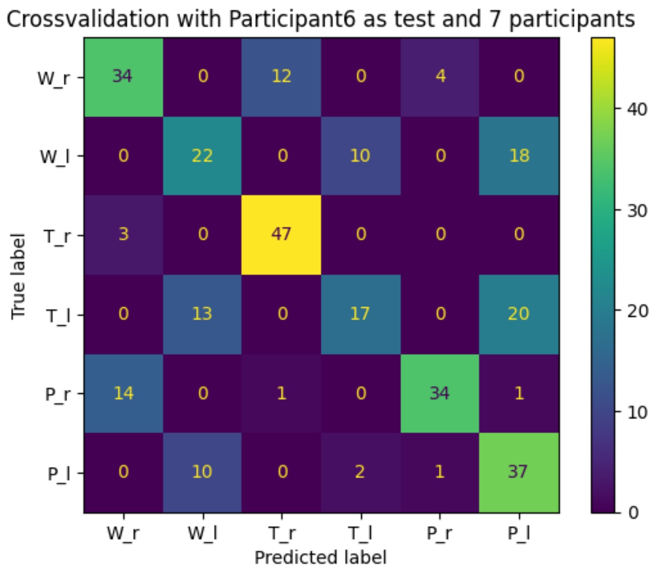


Figure 4.6: Cross Validation with Participant 6 as Testset

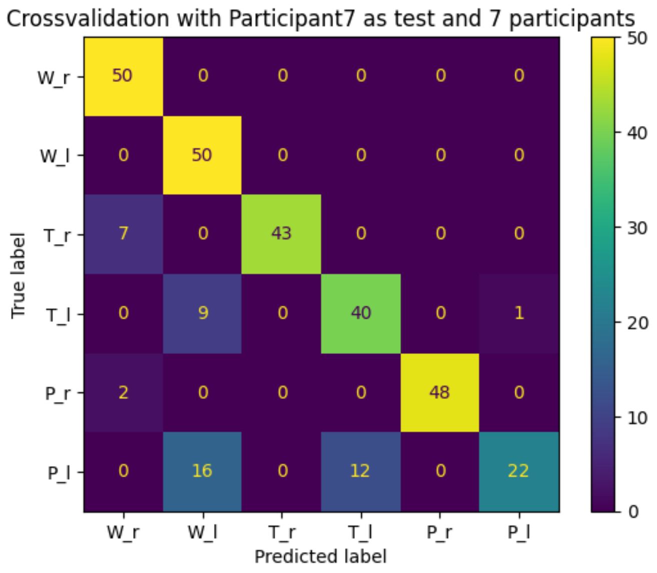


Figure 4.7: Cross Validation with Participant 7 as Testset

5 Future Work

This section will give insight into which further improvements could be made to the project and show what future projects could be pursued.

5.1 Possible Improvements

All in all, the thesis achieved the goal set. It created a solid proof of concept for action recognition in VR and a dataset solely based on information provided by the VR headset and its handheld controllers. As was discussed in the Analysis section, the results could be better. The network could still be extended, and more actions could be added to the dataset. Since this thesis was only a proof of concept, this will have to be achieved in future endeavors.

The first way to improve the Neural Network would be to extend the dataset. As discussed in the Analysis section, when a person does not perform the actions perfectly, the network has trouble recognizing the action. With a more extensive and more diverse dataset, more possible executions could be covered. Through that, the network could learn more diverse movements and different ways of executing the planned movements. Also, in the current dataset, only two out of the seven participants were left-handed. This brings a bias into the dataset that could be remedied by recording more participants. Other human factors, like height, weight, range of motion, and others, are also not perfectly covered by this neural network. More diverse participants could allow for more accurate predictions. The network could also better recognize people's actions that are not in the training set.

Another possible next step, after extending the dataset, would be to expand the size of the Neural Network. With more people in the dataset, the network might need to be bigger to fully analyze and learn the patterns found in the data. The accuracy could also significantly increase with a bigger network and more hyperparameter tuning. A more robust network could be realized through more nodes and layers, which could be extended with more actions.

Due to the easily expandable nature of the scripts, more actions could later be added to the dataset. With more actions, like handshakes, or even full body tracking actions, like running or walking, the network would become even more useful in helping in the modern workspace and with communication in VR. Other more common actions could also be added for more general use cases.

The last way to improve the network would be to try a multi-modal approach. Although this thesis set out to only use the data provided by VR headset and the handheld controllers, as seen in the Related Work section, many papers use more multi-modal approaches. A first approach could be to use video data. Together with the data from the VR headset, a similar setup to the paper by Li et al. [21] can be achieved. By extracting the data from the recorded video and the VR headset, a certain degree of independence from the camera can be achieved. Another multi-modal approach could be to use sensors, like gyroscopes or accelerometers, to support the data provided by the VR system. A combination of sensors, videos, and VR systems is also feasible. This would remedy the problems that each system has on its own. For example, the sensor and VR data would remedy the problem of not being view-independent, which is caused by the camera.

5.2 Possible Future Extensions

As outlined in the paper's beginning, this research aimed to create a basis for human-robot interaction. Once the network has been tweaked to an acceptable level, the first step could be to create a functional gym scene. The scene could contain objects the user can interact with, like levers or buttons. It could also contain objects that can be picked up, like tools or other utensils. A digital representation of a robot could be created for this gym scene. The Neural Network could be used to recognize which actions a user is performing, such as a pointing action towards a tool. The output could be relayed to the digital robot. The distance between the tool and the user could be determined through the information provided by the scene. The digital robot could then determine if the tool needs to be brought closer to the user. This would be a simple proof-of-concept setup for human-robot interactions in the VR space.

The virtual environment mentioned in the last paragraph could also be used in a more work-related setting, assuming a few tweaks are made. The robot could provide information about the current workspace through RGB and depth cameras. Through this, a virtual representation could be created, similar to the digital twin mentioned by Dallel et al. [10]. The RGB video could be directly passed through to the VR headset in this environment. This would enable the user to use the robot as a helper: for example, to bring tools or fetch items that are too far away. The robot could also help hold up heavy objects. This could be triggered by the user raising their hand and pointing at the object to be lifted. Another approach could be to use the robot in harmful environments to humans. The user could be stationed in a secure environment and control the robot from afar. In this approach, the robot could mimic the recognized actions the user performs. The joystick on one of the VR handheld controllers could control the robot's position, while all actions performed with the robot's "arms" could mimic the user's behavior. For example, hand raises could be directly translated to raising the robot's arms, and the throwing action could be helpful if objects have to be thrown.

The network could also be used in virtual communication. Since the tracking of the handheld controllers on VR headsets is not always perfect, especially when using inside-out tracking, the network could substitute information. When presented with a virtual avatar, the avatar's animations could be played based on the network's predictions. More natural communication could be achieved when communicating in a VR space with another person who is also in the VR space. For example, when a user performs a wave motion towards another user, the controller might be out of tracking range shortly. This could lead to the model potentially glitching. If the network output was also used, an animation of the avatar waving could be played. This could help with the immersion in virtual communication but should be explored separately in other work.

Another possible improvement to this thesis could be to fully flesh out the given prototype and turn it into a bigger project. A fully-fledged UI could be created to display the actions that have just been performed by the user in the virtual reality space. It could also be built to run natively on different standalone headsets, allowing for greater portability. Since different headsets, like the Valve Index, can provide more information, different versions could be created for different headsets to best utilize their entire functionality. A balance between portability and feature set would have to be found for each device.

6 Conclusion

This thesis aims to construct a dataset and propose a proof of concept for a machine-learning approach regarding action recognition in VR. The dataset was chosen to be three actions with a differentiation between left and right. The actions chosen were throwing, waving, and pointing. A Unity script was created that sampled the position, velocity, and rotation of the handheld controllers and the VR headset with a frequency of 30 Hz. The recorded information was then saved to a CSV file. Seven participants recorded actions for the dataset. At first, one of the participants, already in the training set, also recorded the test dataset, but later, the "leave-one-subject-out" approach was used instead. With this approach, one participant was selected as the test set while the others were used as the training set. This was repeated for each participant.

The machine learning approach chosen was a Neural Network due to its common usage in recognition and classification tasks. Due to its ability to recognize spatial and temporal contexts, an LSTM layer was used in the Neural Network. The network was constructed and trained in the Google Colab Cloud environment using an Nvidia T4 GPU. Multiple tweaks were made to increase the network accuracy. After multiple iterations of different dataset sizes and hyperparameter tuning, the network achieved a peak accuracy of 97%, using the "leave-one-subject-out" cross-validation technique. Although this is an excellent result, improvements could be made to the dataset and the network. The dataset could be extended by adding more actions and extending it to include more people. The network could also be expanded upon with more and larger layers. Further research could be done on using multi-modal approaches.

A Unity script was created that uses the trained Neural Network to perform live predictions in the VR space. The performed action is sent to the network in Unity, and the prediction is displayed in the console. This script can later be used as a basis and expanded on by other researchers. It could also be ported to other game engines.

This thesis creates an excellent proof of concept for VR-based action recognition that only uses information provided by the VR headset and handheld controllers. It lays the foundation for future work in the direction of using VR-based action recognition to perform tasks in dangerous working environments, as well as for communication in a VR space.

List of Figures

Bibliography

[1] A. F. Agarap. "Deep learning using rectified linear units (relu)." In: *arXiv preprint arXiv:1803.08375* (2018).

[2] L. P. Berg and J. M. Vance. "Industry use of virtual reality in product design and manufacturing: a survey." In: *Virtual reality* 21 (2017), pp. 1–17.

[3] R. Bogue. "The growing use of robots by the aerospace industry." In: *Industrial Robot: An International Journal* 45.6 (2018), pp. 705–709.

[4] J. Brownlee. *LSTMs for human activity recognition time series classification*. 2020. URL: `https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/` (visited on 04/01/2024).

[5] G. Corporation. *Google Vertex AI*. URL: `https://cloud.google.com/vertex-ai` (visited on 04/03/2024).

[6] U. Corporation. *Barracuda*. 2018. URL: `https://docs.unity3d.com/Packages/com.unity.barracuda@3.0/manual/index.html` (visited on 03/30/2024).

[7] U. Corporation. *OpenVR*. 2018. URL: `https://docs.unity3d.com/2018.4/Documentation/Manual/VRDevices-OpenVR.html` (visited on 03/30/2024).

[8] V. Corporation. *Steam VR*. 2019. URL: `https://www.steamvr.com/` (visited on 03/30/2024).

[9] V. Corporation. *Valve index® headset on steam*. 2019. URL: `https://store.steampowered.com/app/1059530/Valve_Index_VRHeadset/` (visited on 03/22/2024).

[10] M. Dallel, V. Havard, Y. Dupuis, and D. Baudry. "Digital twin of an industrial workstation: A novel method of an auto-labeled data generator using virtual reality for human action recognition in the context of human–robot collaboration." In: *Engineering applications of artificial intelligence* 118 (2023), p. 105655.

[11] T. Developers. "TensorFlow." In: *Zenodo* (2022).

[12] M. Dwarampudi and N. Reddy. "Effects of padding on LSTMs and CNNs." In: *arXiv preprint arXiv:1903.07288* (2019).

[13] P. Elias, J. Sedmidubsky, and P. Zezula. "Understanding the limits of 2D skeletons for action recognition." In: *Multimedia Systems* 27.3 (2021), pp. 547–561.

[14] A. T. Goh. "Back-propagation neural networks for modeling complex systems." In: *Artificial intelligence in engineering* 9.3 (1995), pp. 143–151.

[15] M. Heydarian, T. E. Doyle, and R. Samavi. "MLCM: Multi-label confusion matrix." In: *IEEE Access* 10 (2022), pp. 19083–19095.

[16] K. Hirota and T. Komuro. "Grasping Action Recognition in VR Environment using Object Shape and Position Information." In: *2021 IEEE International Conference on Consumer Electronics (ICCE)*. 2021, pp. 1–2. DOI: 10.1109/ICCE50685.2021.9427608.

[17] M. A. Istiake Sunny, M. M. S. Maswood, and A. G. Alharbi. "Deep Learning-Based Stock Price Prediction Using LSTM and Bi-Directional LSTM Model." In: *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. 2020, pp. 87–92. DOI: 10.1109/NILES50944.2020.9257950.

[18] I. K. M. Jais, A. R. Ismail, and S. Q. Nisa. "Adam optimization algorithm for wide and deep neural network." In: *Knowl. Eng. Data Sci.* 2.1 (2019), pp. 41–46.

[19] S. Ji, W. Xu, M. Yang, and K. Yu. "3D Convolutional Neural Networks for Human Action Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1 (2013), pp. 221–231. DOI: 10.1109/TPAMI.2012.59.

[20] H.-T. Li, Y.-P. Liu, Y.-K. Chang, and C.-K. Chiang. "Action recognition and tracking via deep representation extraction and motion bases learning." In: *Multimedia Tools and Applications* 81.9 (2022), pp. 11845–11864.

[21] X. Li, H. Chen, S. He, X. Chen, S. Dong, P. Yan, and B. Fang. "Action recognition based on multimode fusion for VR online platform." In: *Virtual Reality* 27.3 (2023), pp. 1797–1812.

[22] C. Liang, J. Lu, and W. Q. Yan. "Human action recognition from digital videos based on deep learning." In: *Proceedings of the 5th International Conference on Control and Computer Vision*. 2022, pp. 150–155.

[23] C. Liu, Y. Hu, Y. Li, S. Song, and J. Liu. "Pku-mmd: A large scale benchmark for continuous multi-modal human action understanding." In: *arXiv preprint arXiv:1703.07475* (2017).

[24] A. Logacjov, K. Bach, A. Kongsvold, H. B. Bårdstu, and P. J. Mork. "HARTH: a human activity recognition dataset for machine learning." In: *Sensors* 21.23 (2021), p. 7853.

[25] P. I. P. Ltd. *Pico Link*. 2022. URL: https://www.picoxr.com/de/software/pico-link (visited on 03/30/2024).

[26] P. I. P. Ltd. *Pico4*. 2022. URL: https://www.picoxr.com/de/products/pico4 (visited on 03/30/2024).

[27] J. Lu, M. Nguyen, and W. Q. Yan. "Deep Learning Methods for Human Behavior Recognition." In: *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)*. 2020, pp. 1–6. DOI: 10.1109/IVCNZ51579.2020.9290640.

[28] R. Mutegeki and D. S. Han. "A CNN-LSTM Approach to Human Activity Recognition." In: *2020 International Conference on Artificial Intelligence in Information and

Communication (ICAIIC). 2020, pp. 362–366. DOI: 10.1109/ICAIIC48513.2020.9065078.

[29] Neuralthreads. *Categorical cross-entropy loss*. 2021. URL: https://neuralthreads.medium.com/categorical-cross-entropy-loss-the-most-important-loss-function-d3792151d05b (visited on 04/01/2024).

[30] Onnx. *Onnx/tensorflow-onnx: Convert tensorflow, Keras, tensorflow.js and tflite models to ONNX*. 2019. URL: https://github.com/onnx/tensorflow-onnx.

[31] M. Schröder and H. Ritter. "Deep learning for action recognition in augmented reality assistance systems." In: *ACM SIGGRAPH 2017 Posters*. 2017, pp. 1–2.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[33] X. Wang, J. Zhang, and W. Q. Yan. "Gait recognition using multichannel convolution neural networks." In: *Neural computing and applications* 32.18 (2020), pp. 14275–14285.

[34] Y.-c. Wu and J.-w. Feng. "Development and application of artificial neural network." In: *Wireless Personal Communications* 102 (2018), pp. 1645–1656.

[35] S. Yan, Y. Xiong, and D. Lin. "Spatial temporal graph convolutional networks for skeleton-based action recognition." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[36] L. Ye and S. Ye. "Deep learning for skeleton-based action recognition." In: *Journal of Physics: Conference Series*. Vol. 1883. 1. IOP Publishing. 2021, p. 012174.