TU WIEN Informatics

Attention Based Neural Network for Autonomous Driving Agents

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Stefan Ulmer, MSc (WU)
Matrikelnummer 11777750

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Head Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Mitwirkung: Univ.Ass. Dott.mag. Luigi Berducci

Wien, 25. August 2023

Stefan Ulmer	Radu Grosu

TU WIEN Informatics

Attention Based Neural Network for Autonomous Driving Agents

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Stefan Ulmer, MSc (WU)
Registration Number 11777750

to the Faculty of Informatics

at the TU Wien

Advisor: Head Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Assistance: Univ.Ass. Dott.mag. Luigi Berducci

Vienna, 25th August, 2023

Stefan Ulmer	Radu Grosu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Stefan Ulmer, MSc (WU)

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. August 2023

Stefan Ulmer

Danksagung

Allem voran bin ich dankbar für meine Studienzeit an der TU Wien, es war eine sehr aufregende Zeit, die mir immer in Erinnerung bleiben wird. Ich bin auch dankbar für die vielen Freunde, die ich während meiner Zeit an der TU Wien gewinnen durfte. Weiters bedanke ich mich bei den beeindruckenden Persönlichkeiten in der Lehre, die mich inspiriert und mein Wissen erweitert haben.

Diesbezüglich will ich mich auch bei allen Personen bedanken, die zur Umsetzung meiner Diplomarbeit beigetragen haben. In erster Linie will ich mich bei Radu Grosu und Luigi Berducci bedanken, die mich während meiner Arbeit großartig ünterstützt haben. Sie haben mich bei technischen und fachlichen Fragen, als auch mit hilfreichen Vorschlägen unterstützt. Ich will mich auch bei meiner Ehefrau bedanken, die immer Verständnis gezeigt hat, auch wenn dies während der Zeit meines Studiums nicht einfach war.

Acknowledgements

First, I am grateful for my study time at TU Vienna. I will always remember myself in this very exciting and challenging period. I am also thankful for the many friends I made during my time at TU Vienna. Furthermore, I will thank the impressive personalities of the teaching staff who inspired me and expanded my knowledge.

In this regard, I want to express my gratitude to all the individuals who contributed to the realization of my thesis. Firstly, I want to thank Radu Grosu and Luigi Berducci, who provided tremendous support during my work. They assisted me with technical and professional questions and offered valuable suggestions. I also want to thank my wife, who has always been understanding, even when it was not easy during my studies.

Kurzfassung

In dem letzten Jahrzehnt hat autonomes Fahren kontinuierlich an Innovation zugelegt und die Sicherheit verbessert. Von anfänglichen Tests im Labor, schaffte es die Automobilindustrie, das autonome Fahren auf öffentliche Straßen zu bringen und die Forschung intelligenter Systeme voranzutreiben. Das Ziel ist, Verkehrsunfälle zu reduzieren und die menschliche Mobilität zu erhöhen. Der technologische Fortschritt hat die Zuverlässigkeit und Leistung der Hardware für das autonome Fahren verbessert, trotzdem haben diese Systeme Schwierigkeiten, alle Szenarien der realen Welt zu meistern. Eine Lösung für dieses Problem ist maschinelles Lernen, da es aus vorhandenen Fahrszenarien lernt, wie vergleichbare Situationen bewältigt werden können. Dies hat die Forschung in maschinellem Lernen vorangetrieben, in den letzten Jahren wurde untersucht wie machinelles Lernen das autonome Fahren unterstützen kann. Obwohl es heutzutage noch kein vollständig autonomes Fahrsystem gibt, das mit der Komplexität des realen Straßenverkehrs umgehen kann, machen viele Systeme rasante Fortschritte in Richtung vollständiger Autonomie. Trotz der Verbreitung des Begriffs *selbstfahrend*, gilt es spezielle Kategorien hinsichtlich des Autonomie-Levels zu berücksichtigen. Ein Fahrzeug muss ein bestimmtes Level an Autonomie erreichen, um diese autonomen Systeme im realen Verkehr bereitstellen zu dürfen.

Wertvolle Erkenntnisse in Bezug auf das Fahrverhalten sind durch *menschliche Aufmerksamkeit* erlangbar, diese Aufmerksamkeit spielt eine wichtige Rolle bei der Identifizierung und Reaktion auf verschiedene Straßenbedingungen und potenzielle Gefahren. Um die Fähigkeiten bestehender autonomer Fahrsysteme weiterzuentwickeln, können etwaige Vorteile durch die Berücksichtigung der menschlichen Aufmerksamkeit in den Trainingsprozess einbezogen werden. Durch die Beachtung der menschlichen Aufmerksamkeit als zusätzliches Merkmal für das Training neuronaler Netzwerke, berücksichtigt diese Masterarbeit Verbesserungen hinsichtlich der Vorhersage des Lenkwinkels eines autonomen Fahrsystems auf einer Strecke.

Für die Verwendung von maschinellem Lernen musste ein Datensatz mit Fahrszenarien erstellt werden, welcher die dazugehörigen menschlichen Aufmerksamkeitsbereiche inkludiert. Das F1tenth Framework wurde verwendet, um das Auto auf der Strecke zu fahren und mittels montierter Logitech Kamera wurde ein Video aufgezeichnet. Der Datensatz ist bildbasiert, wobei für jedes Bild aus dem Video der menschliche Aufmerksamkeitsbereich mit der ViewPointSystem-Brille extrahiert wurde, um die entsprechenden

Blickkoordinaten im Datensatz zu speichern. Es wurde ein neuronales Netz implementiert, dass mit dem Datensatz trainiert wurde, um die menschlichen Aufmerksamkeitsbereiche vorherzusagen. Im Anschluss musste ein neuronales Netzwerk designed werden, das einen Lenkwinkel auf Basis eines eingehenden Kamerabildes vorgibt. Dieses Bild enthält einen markierten Bereich, welcher den menschlichen Aufmerksamkeitsbereich repräsentiert. Nach adäquater Funktion in Simulation, wurde der gesamte Workflow auf einem F1tenth Auto eingesetzt, um die Funktionsweise auf einer echten Strecke zu testen. Sowohl die Ergebnisse in der Simulation, als auch auf einer realen Strecke, sind für den verwendeten Datensatz angemessen. Daher deutet dies auf das Potenzial hin, die Aufmerksamkeit eines Menschen als zusätzliches Merkmal in zukünftige autonome Fahrsysteme zu integrieren.

Abstract

Over the last decade, autonomous driving has continuously evolved with respect to innovation and safety. From the tests in a laboratory at the early stage of development, the automotive industry managed to get the cars to public roads, advancing the research of intelligent systems. The overall aim is to reduce traffic accidents and increase the possible mobility for people. The technological advance improved the reliability and performance of the hardware used for autonomous driving, but these systems have struggled in adapting to the complexity and variety of scenarios in the real world till recent years. Machine learning proved to be an effective method to enhance the adaptation of these systems to a broad variety of driving scenarios, through the learning from historical data and the generalization capabilities of deep-learning models. This pushed the research in machine learning topics and a lot of research was done in the last recent years on how machine learning can support autonomous driving systems. Nowadays, there is no fully autonomous driving system that is able to deal with the complexity of real-world driving, but many systems are fastly progressing towards full autonomy. Despite the wide adoption of *self-driving* terminology, there are many categories for classifying the autonomy level. A car has to fulfill a certain level for the allowance to provide these autonomous systems in real traffic.

Valuable insights into driving behavior are provided by *human-attention*, a crucial element in driving, playing a significant role in identifying and reacting to various road conditions and potential hazards in driving situations. To advance the capability of existing autonomous driving systems, the potential benefits of incorporating human-attention can be considered for the training process. By including human-attention as an additional feature for training neural networks, this master thesis observes improvements regarding the steering angle prediction of an autonomous driving system on a track.

For the usage of machine learning, a dataset with driving scenarios including human-attention had to be created. The F1tenth framework was used to operate the car on the track and a mounted Logitech camera on the car recorded a video during driving to create the frame-based dataset. Additionally, the ViewPointSystem-glasses were used to record the human-attention for the specific frames and save the corresponding attention-gaze coordinates for the dataset. From the dataset as starting point an attention model was implemented and trained to predict the human-attention for incoming frames. When this prediction was reasonable, the next step was the design of an agent model

that would predict a steering angle based on an input frame with a human-attention area marked on it. After having evaluated the performance of the trained agents in simulation, the deployment of the trained model was done on a F1Tenth racecar. The results in simulation and real-world deployment show encouraging improvements due to the use of human-attention features. Therefore, it suggests the potential of integrating human-attention models in future autonomous driving systems.

Contents

CHAPTER 1

Introduction

The importance of autonomous driving nowadays is substantial and a good reliance on these systems is a fundamental criterion for the users. Continuously more human-driving actions will be outsourced to these systems. Therefore, consistent research activity is necessary for an ongoing improvement of all these systems to reach an adequate driving performance considering safety as a top priority. A human's cognitive focus-ability can be beneficial for driving scenarios and this factor can be considered to improve the safety of driving agents. This thesis focus on the use of *human-attention* to enhance *autonomous driving*. In particular, we collect a considerable amount of training data from real-world interaction and leverage Machine Learning (ML) models to autonomously control a vehicle.

In the following, we motivate our work in Section 1.1, we present the proposed workflow considering the *human-attention* feature in Figure 1.1. We discuss the details for a possible solution-delivery in Section 1.3.

1.1 Motivation and Problem Statement

Autonomous driving is an important and impactful process that is revolutionizing the automotive industry today. The need for autonomous driving arises from the goal of reducing the likelihood of car accidents caused by human factors (e.g., drowsiness, fatigue, inexperience, aging). From a safety and traffic perspective, it would be most favorable to use level-5 [1] fully-autonomous vehicles to reach that goal.

Different types of agents are commonly used for the implementation of autonomous driving algorithms. Starting from simple reflex to more complex goal-based agents they all have to consider the current state of the agent's environment, to take an appropriate action, possibly in a probabilistic fashion. Based on a given state, the agent can take an action that leads to the following state [2].

During the last decade, the ongoing developments in Deep Learning (DL) and Artificial Intelligence (AI) lead to boosts in computer vision and robotics. An important milestone was the Defense Advanced Research Projects Agency (DARPA) challenges in the years 2004-2007, intending to complete an off-road track as the fastest vehicle. In the challenge of 2005, the winning team used ML-techniques and this introduced the acknowledgment of AI-components in autonomous driving. Associated with these advancements, autonomous driving applications continuously evolved in the aspect of innovation and safety. In the beginning, the progress started with cars in the laboratories, but with growing success, they managed to bring testing to public roads. These systems are capable of reducing accidents and increasing general mobility, bringing up *self-driving* performance, but there is still a classification necessary regarding the different autonomous levels [3].

The usage of an increasing number of sensors and actuators as well as deep neural networks for a car's automatic action are necessary to provide safe and stable driving. Implementing *single agents* with good decision-making capabilities is hard, but required to reach this. Classical approaches cannot capture all possible upcoming events and incidents, a promising solution is the use of *machine-learning*. Even though reinforcement learning is an emerging area in autonomous driving, the applications that are commercially and successfully used now are manageable. The difficulties for this task arise from the scarce availability of datasets and specific literature [4]. Recent studies include attention as a mechanism to interpret and improve algorithms used for deep learning [5].

In this context, it is hard to synthesize models and algorithms for driving behavior without compromising safety. In virtual reality, attention already shows improvements for existing networks [6]. Since humans can cope with complex driving scenarios, we think that a *human-inspired attention-mechanism* could improve autonomous driving. To this aim, we propose to consider *human-attention to decide a vehicle's control* for a specific state.

In the scope of this thesis, we consider *human-attention* and study possible improvements of autonomous driving by the use of it. In the following, we present each Research Question (RQ) we intend to answer. The starting point has to be an adequate dataset that contains the information for human-attention, the realization of that part considers RQ0. When the whole implementation is done, the results of the agent Neural Network (NN), especially considering improvements by human-attention, have to be analyzed as stated in RQ1. The whole process flow shown in Figure 1.1 has to be deployed on the real F1/10 car to compare the outcome to the simulation (RQ2). The last research question RQ3 aims to consider the performance of the implementation on the F1/10 car for a different track.

RQ0 *How can we obtain a rich enough labeled dataset for human attention in driving situations?*

RQ1 *How much is a neural network's training-and-prediction performance improved by the attention mechanism?*

RQ2 *What differences are observable and measurable between the simulation and real car performance?*

RQ3 *What is the robustness of the trained agent to domain shift (e.g. different track)?*

The content is organized as follows: Chapter 1 introduces the motivation, background, general idea, and the methodological approach; Chapter 2 presents the theoretical foundation of DL; Chapter 3 describes the dataset generation; Chapter 4 emphasizes the neural network for the prediction of human-attention; Chapter 5 presents an agent NN to predict driving commands; Chapter 6 concludes the results of this thesis;

1.2 Related Work

Based on relevant literature, one method for training agent policies to achieve good driving behavior is deep reinforcement learning. This technique has shown effectiveness in producing collision-free behaviors for autonomous driving agents [7, 8]. Autonomous driving often includes additional integrated subsystems: Localization, Perception, Planning [9]. The increased usage of sensors (e.g. LiDAR) and the combination of computer vision and hardware acceleration pushed the improvements of autonomous driving, whereas the overall goals are reliable results in real-time. Recent systems are in continuous development for real traffic environments because current computing systems are still not reliable and robust enough for autonomous driving on level 5 [10, 11].

The preprocessing of data transformations is one of the key necessities for an adequate ML training process. The performance of DL systems is highly dependent on the fact of how well the used dataset is prepared because a NN needs a highlighted relationship in the data to learn the necessary features acceptably. This part of feature engineering for a good dataset is labor-demanding and a common shortcoming of learning approaches. Hence, a strategy for getting a good data representation is representational learning [12]. This approach is used to learn current data representations based on the monitored interpretive input features, to create a fitting data representation and data distribution that enhance the NN to learn the data relationships more easily.

However, Wang and Yang show that unsupervised models present sub-optimal performance. They included attention to their NN architecture, by a combination of supervised learning with a Recurrent Neural Network (RNN), to improve the training process and

the overall performance of the trained model. Furthermore, Vaswani et al. stated that the best-performing models integrate an attention mechanism and came up with a new architecture - *the Transformer*. They proved that in translation tasks, the models need less training time for achieving adequate performance. Makrigiorgos et al. shows that in simulations and virtual reality, these models improve accuracy.

Nevertheless, the existing literature about attention and its usage within a neural network is manageable. Additionally, there is a lack of information about testing these model systems on real cars. Hence, the result of this thesis will show the driving performance of a real F1/10 car [15] based on model predictions that include an attention-mechanism.

1.2.1 Learned Attention

Typical model architectures for sequence modeling adapt RNN and Long Short-Term Memory (LSTM) to learn the dependencies between text sequences. These architectures suffer from scalability to long sequences, as computational power and the memory needed rise as well, bringing a quantity limitation for examples that can be used for the training. The attention-mechanism used for Natural Language Processing (NLP) models stated by Vaswani et al. is favorable to learning the relationships between the input and output, reducing the amount of computation resources needed by significantly more parallelization. The proposed model is called *Transformer* and its encoder-decoder architecture does not use a RNN or Convolutional Neural Network (CNN), instead it only relies on the self-attention to learn the necessary dependencies.

The Transformer uses a *Multi-Head* attention mechanism that put weights on the different sequence positions. The scaled dot-product of attention is used as a mapping for query and key-value pairs, this layer is integrated multiple times into the Multi-Head attention. Based on that it is possible to take the average of the attention-weighted positions, whereas the architecture considers several layers of the scaled dot-product attention and is running in parallel. This mechanism is the reason for the possibility of additional parallelization and a reduction in training time, therefore this self-attention approach was privileged to be used for the Transformer [14]. Furthermore, self-attention is also doing well in other assignments, like summarizing texts in an abstract way or comprehensive reading.

1.2.2 Human-inspired Attention

The *human-attention* used for the dataset generation (see Chapter 3) is different from the self-attention mechanism discussed in Section 1.2.1. The created dataset based on the recordings of human-attention is used for a supervised learning approach for the attention-NN (discussed in Chapter 4). Compared to the self-attention for NLP, the aim of human-inspired attention is a visual representation of human-attention on a real input frame of a camera. The interesting information that is necessary for the

ongoing work on the agent-NN (see Chapter 5) has to be produced as a 2D image. An example is shown in the whole process flow in Figure 1.1, where the forwarded 2D-image after the *Attention-Model* got enhanced with a red marked area that represents the human-attention based on the prediction of the attention-NN.

During the recording process of the dataset, two interesting properties of human-attention were observable:

- Human-attention is *jumpy*
- A correctness check on human-attention is tough

The human-attention was recorded for each incoming frame via the View Point System (VPS)-glasses (see Section 3.1), this system provides the human attention gaze for an image in x and y coordinates. It is noticeable, that human-attention shows the possibility of huge changes in the coordinates-position from one input frame to the next. The detected behavior can be seen as *jumps* in the human-attention. The correctness of human-attention for a specific input frame is hard to tell, during the recording sessions the same human driver can have different coordinate positions of the attention area for the same input frame. Hence, there is not only one correct solution, resulting in multiple human-attention areas that can be valid positions on an image.

1.3 Methodological Approach

The effort of this thesis work can be split up into multiple work packages. Starting from the literature review, machine learning techniques have to be used for the training phase of a proper attention- and agent-model. After checking the model outcomes in the simulation environment, extensive analysis and evaluation will be done on the real F1/10 racecar.

The implementation idea of this complete workflow setup is shown in Figure 1.1, starting with input frames on the left, which have to be recorded to generate a dataset for the ML parts. The *Attention-Model* (see Chapter 4) has to be able to take these frames as input and enhance the images with a red marked area representing the *human-attention*. This frame with the predicted human-attention has to be forwarded to the *Agent-Model* (see Chapter 5). Based on the input, the *Agent-Model* has to be capable of driving command prediction to drive the F1/10 car on the track.

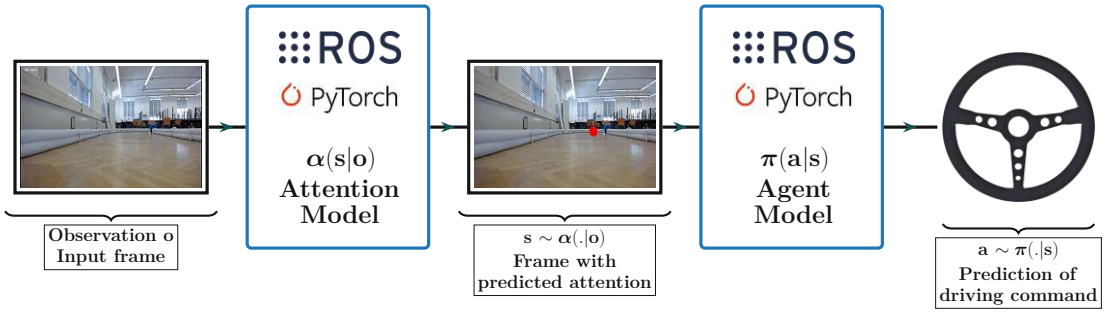


Figure 1.1: Attention-Agent prediction workflow.

1.3.1 Literature Review

For getting the necessary knowledge about autonomous driving and deep learning, a review of the existing literature has to be done beforehand. Furthermore, based on this knowledge the aim is to get a deeper understanding of the F1/10 framework for the simulation and the real car. Especially for Chapter 2, the theoretical aspects of DL have to be considered for a better perspective on the created architectures for both the attention- and the agent-NN.

1.3.2 Modeling

The machine learning process of the models for predicting attention and driving commands uses the created eye-tracking dataset considering human-attention as the ground truth for each frame. By training a neural network on this human attention dataset, a model has to be developed that predicts the attention focus for a specific frame input. For autonomously driving on a track, an agent neural network has to be trained to predict the car's action based on an Human Attention Frame. These models will be used for controlling the car within the driving environment.

1.3.3 Analysis

When simulation and models have reached the desired level of accuracy, it can be considered to drive the real car on a track. The analysis of the racecar performance will lead to additional refinements and possible iterations in simulation to ensure safety on the real track.

1.3.4 Evaluation

After the complete pipeline (see Figure 1.1) was implemented from training to deployment on the real car, the approach has to be evaluated compared to the state of the art. The

evaluation (Chapter 6) phase will compare the performance between the simulation and the real car, the answering of the research questions will be done in the conclusion.

1.4 Setup

This thesis embraces the work of software implementations for simulation and the deployment of created neural networks via Robot Operating System (ROS) nodes on the real F1/10 car [15]. Hence, the observations are done for both the simulation and the real-car performance. The necessary tasks can be divided into the software stack that was used and the hardware to run everything on a real F1/10 car.

1.4.1 Software Stack

The main part of the implementations had to be done before the deployment on the real F1/10 car. The neural networks for the human-attention prediction and the inference of driving commands were implemented and trained first before the experiments and observations were done on the real car. Generally, the software stack embraces the following software frameworks and applications as illustrated in Figure 1.2. The classification is done based on the different main parts of the thesis:

- Dataset Generation
- Machine Learning
- Hardware Deployment

Overall, the number one programming language for all parts was Python. Visible in Figure 1.2, for the *Dataset Generation* ROS was an essential part. The ROS-framework was running on the F1/10 car and was necessary for the users to drive the car on the track. Additionally, ROS-bags were used to store all the information for the dataset creation afterward. Furthermore, *OpenCV* [16] as a computer vision framework was especially needed for the transformations and projections of the dataset creation. This was necessary for the recorded attention coordinates that are based on the original video from the VPS eye-tracking system and had to be transformed to the raw video from the car webcam (visual representation in Figure 3.3). The framework was further used for the preparation of the frames considering the attention area and resizing steps that had to be taken before the frames got redirected to the neural networks for training and evaluation.

The main part of the work was *Machine Learning* as illustrated in the second block of Figure 1.2, dependent on the Dataset Generation. *Pytorch* [17] is a common Python-based machine-learning framework that was necessary for the creation of the NN architecture. The layer structure of the neural network was constructed with this framework. Additionally, the Pytorch framework supports extensive dataset features that are quite necessary

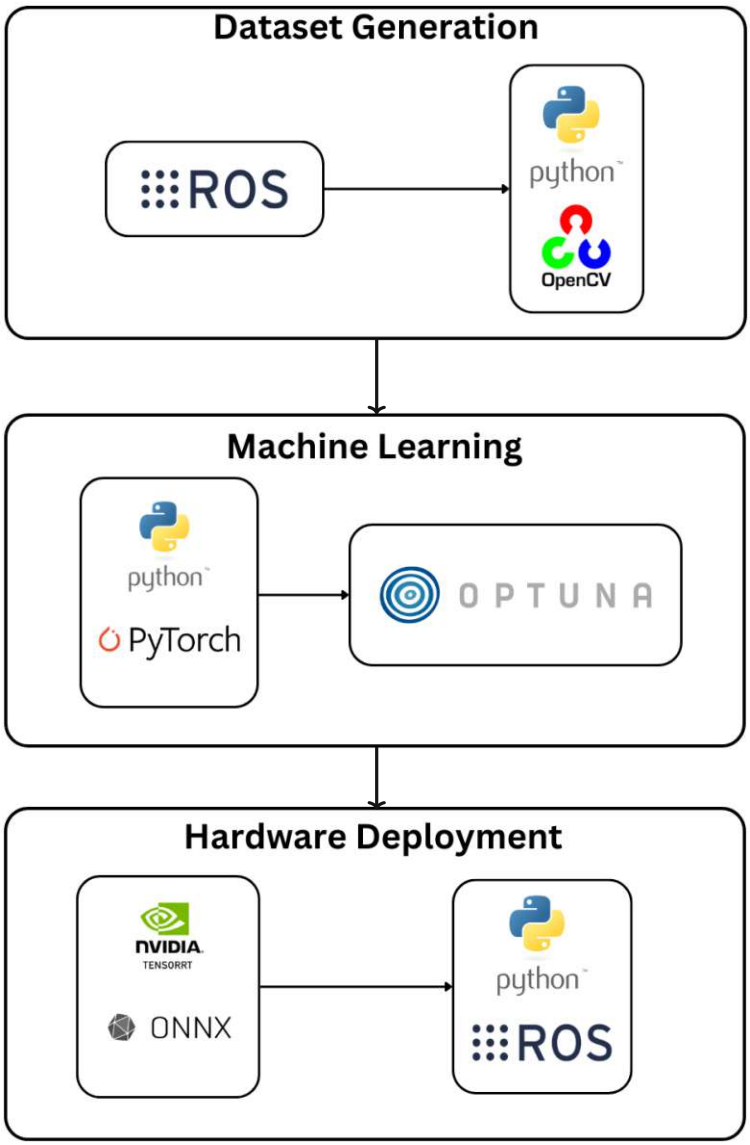


Figure 1.2: Software Stack.

for the preprocessing to provide better data to the models. *Optuna* [18] is a hyperparameter optimization framework for automating the search for the best hyperparameters of a NN training. It was used to find adequate training parameters for the detailed layer architecture of both the attention- and the agent-NN. With this tool, it was possible to get an approximation for the best learning rate and weight decay of the training process.

After the NN implementations the *Hardware Deployment* could be done with the tools

addressed in the last block of Figure 1.2. Regarding a performance upgrade for both neural networks during the inference on the car and the evaluation part, the Pytorch models were exported to the *Open Neural Network Exchange (ONNX)* format. ONNX is a machine learning framework including a model format, in combination with the Python Application Programming Interface (API) of *TensorRT* [19] it was possible to speed up the inference performance by loading the ONNX models with this TensorRT framework. This was especially necessary on the real F1Tenth car because adequate inference performance is essential for reasonable driving commands in time. For the integration on the real F1/10 car, the trained attention- and agent-NN were deployed via the ROS framework in ROS nodes. With these nodes, it is possible to redirect both the predictions of the human-attention and the driving commands via ROS-topics. These topics are used within the F1/10 system [15] for publishing the commands to the controller and operating the car on the track.

1.4.2 Hardware

For machine learning, computing power is essential for adequate performance during the model training. The attention- and the agent-model training are based on input images for feature extraction, therefore the computation work should be done on GPU. The performance-relevant hardware resources used for the machine learning part of this thesis were:

- AMD Ryzen 5 3600 6-Core Processor
- NVIDIA GeForce RTX 3060

Additionally, a TU Vienna server was used for the ability to run several training sessions for speeding up the process. This was also necessary for the evaluation on the car with the implemented Dataset Aggregation (DAGGER)-algorithm discussed in 6.3.2, because NN retraining was necessary on the track and without the server resources it would not have been possible.

The evaluation of model predictions is done in simulation and on a F1/10 car [15]. Based on the general F1/10 build instructions, the car is extended by a Logitech camera. This camera was used for dataset generation (see Chapter 3) and the video/frame-based input for the model predictions. This leads to a F1/10 car version shown in Figure 1.3a.

For the deployment of the real car, the inference of the neural networks has to be done in a reasonable time. Considering the F1/10 build instructions [15], the F1Tenth car for this thesis got assembled a Jetson Xavier NX from Nvidia [20] (see Figure 1.3b). This computer is specially designed for tensor operations, to improve the NN inference performance. The car has been assembled following the documentation of the F1/10 framework [15].

(a) F1/10-car with assembled Logitech camera.

(b) Nvidia Jetson Xavier NX.

Figure 1.3: F1/10-car and assembled Nvidia Jetson Xavier NX.

CHAPTER 2

Deep Learning

This chapter gives a general overview of the theoretical network structures and approaches in deep learning, additionally, this chapter considers all the things that were used for the construction of the attention- and the agent-model. The theory is essential for the ongoing discussions in the following chapters and should help to strengthen the understanding of the machine learning aspects. We will discuss the implementation details for the attention-NN (see Chapter 4) and agent-NN (see Chapter 5), creating proficient models for the prediction of human-attention and driving commands.

First, this chapter gives an introduction to the simplest structures that are necessary to build a neural network. The idea is to explain a neuron and aggregated structures that are formed into layers, with the increasing complexity of these layer structures we will reach the concept of a full neural network architecture.

Furthermore, we will examine two of the most common structures in DL, CNNs (see Section 2.1.1) and RNNs (see Section 2.1.2). A CNN architecture is used for the feature extraction of input images that are forwarded to the neural network structure. Multiple convolution layers can be stacked and use the spatial information of an image by processing it through the layer architecture. Between the layers, there are activation functions included, as well as dropout layers (see section 2.4) to increase the robustness of a neural network [3].

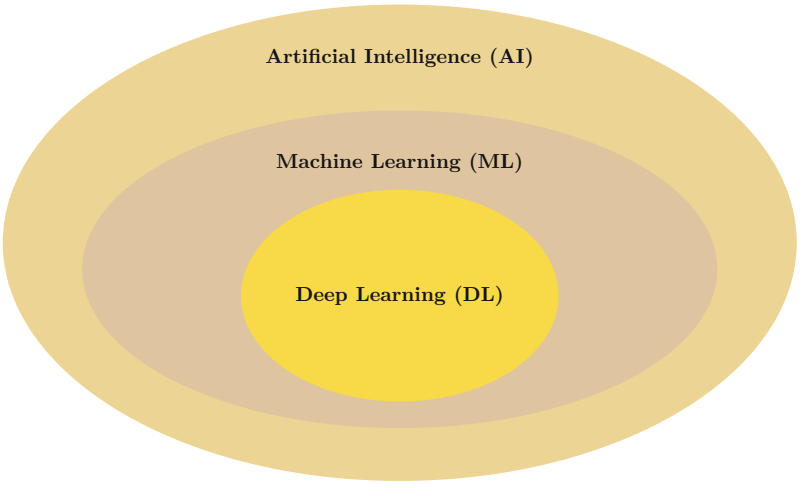


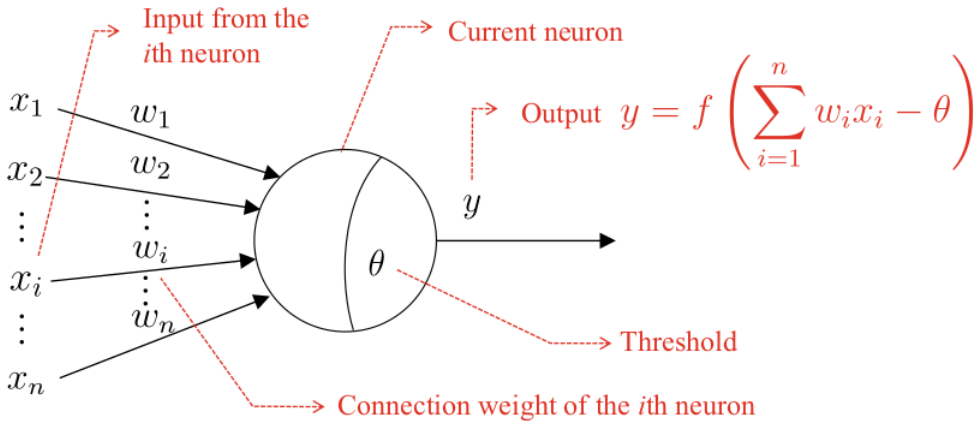
Figure 2.1: Deep Learning classification adapted from [21].

When it comes to DL and learning techniques in general there are different learning denominations. The following classification advertised in Figure 2.1 should help to better distinguish the nomenclature. With respect to software systems that include decisions based on neural networks, these terms are often used synonymously. It is apparent as a top-down hierarchy starting from AI that is a broad consideration of intelligence machine or system behavior. ML therefore can be seen as part of AI because the specific methods used for creating learning-based approaches use data or experience. The next classification is DL as part of ML that uses the same learning approaches based on data but with neural networks that have a multiple-layer architecture. Therefore, DL is recognized as smart AI, *learning* from data (relatable to data science) to uncover insights in data for intelligent decision-making [21].

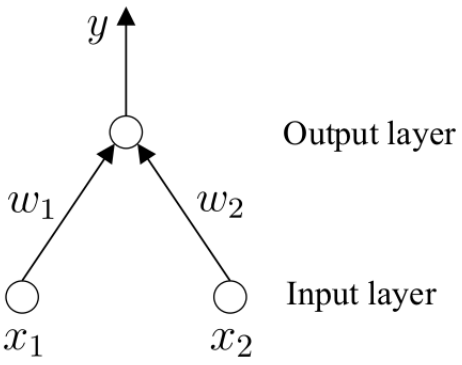
2.1 Neural Network

A neuron is the simplest part of a neural network, in a biological consideration the neuron will be activated by electrical potential when a specific threshold is exceeded. The neuron represented in Figure 2.2a is the so-called *McCulloch and Pitts (M-P) neuron model* that got developed based on the biological neuron. The neuron model considers multiple neurons and weighs each neuron connection, an output signal is triggered if the sum of all input neurons exceeds the threshold. The perceptron consists of two neuron layers (see Figure 2.2b), and logic operations can be executed by this binary classifier based on the values of the input neurons. The possible operations are *AND*, *OR* and *NOT*, the received input signals are calculated based on the chosen operation and forwarded to the

output layer which is a M-P neuron [22].



(a) M-P neuron model.



(b) Perceptron with two input neurons.

Figure 2.2: M-P neuron model and perceptron from [22].

A huge amount of these neurons can be aggregated into a layer, and the connection of several layers will lead to NN-structures with multiple layers displayed in Figure 2.3. In this structure, the neurons are represented fully connected, because each neuron has a connection with all the neurons of the following layer. The neurons are not interconnected with other neurons of the same layer. The NN-models addressed in Figure 2.3 are multi-layer neural networks, the left one has a single hidden layer and the right one serves a deeper version with two hidden layers. The connections between the neurons are essential because during the training these weights are updated and this is what is referred to as the *knowledge* a model was capable to learn. In the same way, the thresholds are updated during the training process, but only for the functional neurons (all neurons after the input layer) [22].

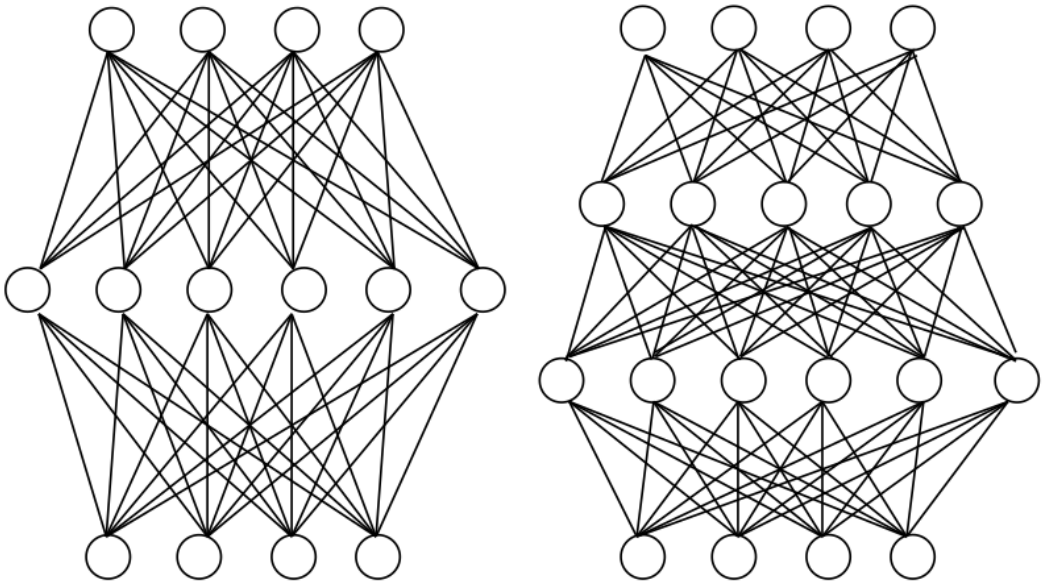


Figure 2.3: Multi-layer feedforward neural network from [22].

A predictor within an artificial neural network is used to map the weighted input to a predicted labeled output. It is possible that a single hidden layer is enough to reach a high accuracy for any given map in an artificial NN. However, DL methods emphasize the usage of multiple layers with few neurons for better computational efficiency than for a single layer with numerous neurons. The effective representation and evaluation of hypothesis maps via parallel and distributed computing infrastructure lead to the success of modern ML methods. In addition, the scalability of the composition of a multi-layer artificial NN enables back-propagation in combination with an effective gradient computation of the loss function [23].

Considering this NN architecture, the main building parts for ML are a *Model*, *Data*, and a *Loss* from training. The process flow of the learning procedure is shown in Figure 2.4, starting with the *model* which does predictions based on an implemented hypothesis. The observation of the predictions will consider the baseline dataset that is used for the validation of the predicted values. The calculated loss based on these predictions is taken into consideration to enhance the inference process for making better predictions. Depending on the prediction accuracy of the model's outcome, the model's training process will be continuously adapted for a prediction improvement [23].

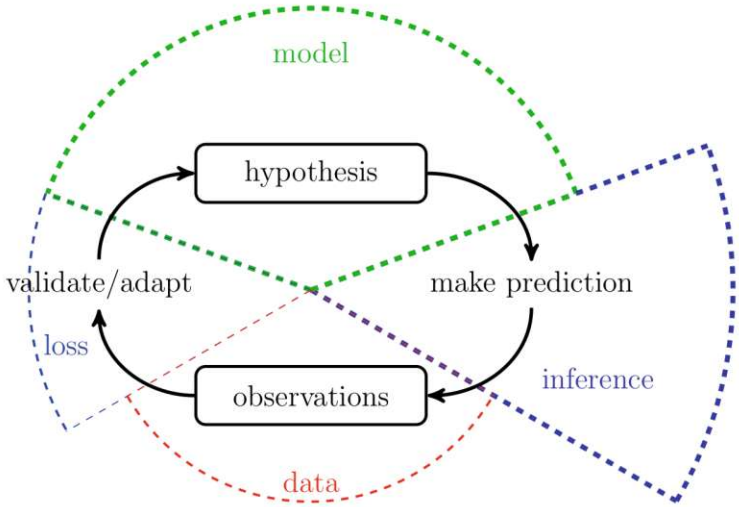


Figure 2.4: Main components of ML from [23].

2.1.1 Convolutional Neural Network (CNN)

Acknowledged by Li et al. CNNs are an essential part of the deep learning area. This type of neural network made an enormous positive impact on several fields of technology e.g. autonomous vehicles, medical analysis, face recognition, and object detection. The multiple-layer architecture including several hidden layers made it possible for artificial neural networks to learn the essential features from the training data.

Previously features had to be extracted by other procedures like local binary patterns or histograms of oriented gradients that were used for computer vision systems. A CNN is automatically learning a feature-space representation based on the training data rather than a handcrafted procedure implementation [3].

As part of DL a CNN has a multiple-layer architecture that is capable of learning the necessary features from the input. With convolution and pooling layers included in the network structure, there is no necessity for human feature extraction. In comparison to a classical Artificial Neural Network (ANN) such as a Multi-Layer Perceptron (MLP), the CNN is an extension with several improvements. It can handle overfitting with dropout and reduces the complexity of the architecture. Every tier employs the most suitable parameters for the training iterations, a CNN layer structure with multiple convolutions and pooling layers is depicted in Figure 2.5. Detailed specified modifications of CNN structures created a lot of popular neural networks, just to name a few examples [21]:

- ◇ Visual Geometry Group (VGG)
- ◇ AlexNet
- ◇ ResNet

For the construction of a CNN several steps have to be processed during the workflow,

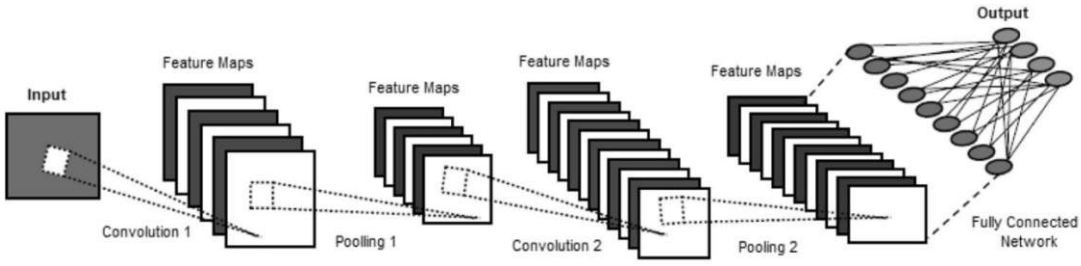


Figure 2.5: CNN with multiple layers from Sarker.

the whole CNN procedure on 2D input is illustrated in Figure 2.6. The feature extraction is based on convolution and results in feature maps, that represent the captured focus during training. A kernel has to be specified that is used like a filter on the input data, additionally, some options have to be set for the process of this iteration. The filtering with a specific *kernel size* will lose information at the border, therefore it is possible to specify *padding* on the input data to adjust the size. The kernel form is moving over the input data to create the feature maps, *stride* is the parameter that specifies the pixel-based step-size for the movement of the kernel. The created feature maps are composed of a huge amount of features and this tends to cause overfitting, to handle this issue max-pooling is used to down-sample the feature maps and avoid redundancies [24].

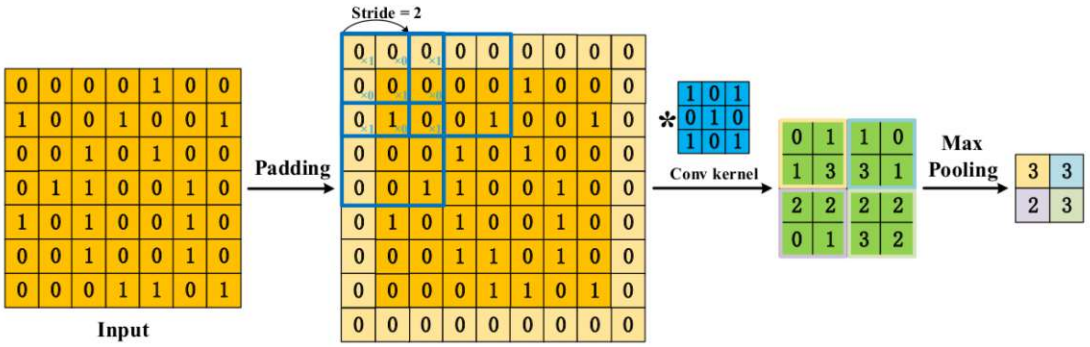


Figure 2.6: CNN procedure from [24].

2.1.2 Recurrent Neural Network (RNN)

When discussing deep learning neural networks, a RNN can be used to consider temporal information of sequenced data [3].

RNNs are favored for sequenced data input in neural networks because these structures are not only providing output but also taking the previous output and forwarding it to the current stage as input, leading to consideration for the ongoing prediction within the

network. In the same way as a CNN, the RNN is learning based on the available training data, but the additional *memory*-part handles the output based on current input with the usage of perceived information from previous steps [21].

Figure 2.7 advertises a comparison of the RNN shown on the left side and the classical feed-forward NN that we discussed in Section 2.1. The RNN contains a loop for the neurons in the hidden layer, representing the recurrent fed back of the generated output again to the same layer as input. The main advantages of a RNN are the additional information storage within the connections and the activations due to the feedback loop of the neurons. Furthermore, it is possible to forward a sequence of data to a RNN, due to the recurrent architecture it can consider e.g. a sentence to an image as input and the learning process has more recognition-features [25].

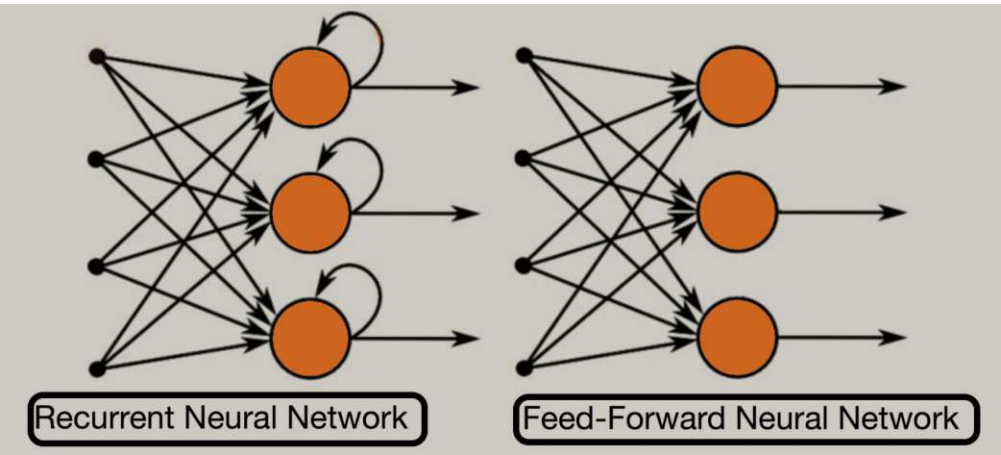


Figure 2.7: Recurrent Neural Network from [26].

A typical problem occurring for RNN are *vanishing and exploding gradients* [27], this issue arises more likely for neural networks with a deeper layer structure. The resulting gradients have to be back-propagated through the network, as this operation considers multiplication with the weights of all the connections in the layer structure, two possible issues reduce the quality of the NN performance. If the resulting values are really small, the vanishing gradients avoid that the NN will learn anything reasonable during the training. If the gradients are exploding, the weights will be oscillating which is also resulting in a bad NN performance [27]. A reasonable successor of RNN that is capable of addressing this issue is the LSTM we will discuss in the following Section 2.1.3, with this layer structure the learning process can be improved.

2.1.3 Long Short-Term Memory (LSTM)

A LSTM is a particular development of RNN and is better able to learn the underlying relationship, the original implementation of the LSTM was done by Hochreiter and Schmidhuber.

As described in Section 2.1.2 the vanishing gradient problem for RNN is a big issue for an adequate NN performance, arising from the fact that the RNN uses a single *tanh*-layer as repeating part in the NN. In difference to the RNN, the LSTM has implemented a layer structure with higher complexity, based on three gates including *sigmoid* layers with pointwise multiplication to decide on changing the cell state [29].

The following 3 gates are necessary for the LSTM implementation [21]:

- ◇ Input-gate
- ◇ Output-gate
- ◇ Forget-gate

The *Input*-gate decides on the information that will be entered into the cell state and the *Output*-gate manages the outputs of the cell. The *Forget*-gate contains the selection part of the stored information, accordingly, it removes information that is no longer useful and memorizes the information that is suitable from the previous state cell. Sarker states that data storage for long periods is done with the memory cell and the information flow of the cell is handled by these three gates which makes the LSTM to a fortunate RNN.

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t]) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t]) \\ \tilde{C}_t &= tanh(W_c \cdot [h_{t-1}, x_t]) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \end{aligned} \tag{2.1}$$

2.1: LSTM gate equations.

The corresponding equations for the following description of the LSTM workflow are addressed in Equation 2.1, where W represents the weight matrix of the vector multiplication for each gate. The procedure for the working gates within the LSTM in Figure 2.8 starts with the *Forget*-gate. It looks at the previous output h_{t-1} and the current input x_t and computes an output between 0 and 1. This output represents the amount of information that will be kept by the cell state because this will be used as a weight for the multiplication with the cell-state C_{t-1}. The *Input*-gate creates a possible cell candidate $\tilde{C}_t$ via the tanh-layer. As the old cell state already had been multiplied with f_t to only keep the relevant information, the new information has to be added by multiplication of i_t (output of the σ-function) with the candidate $\tilde{C}_t$, resulting in an update for the cell-state C_t. The *Output*-gate is based on the cell-state C_t, which is put through the

tanh-layer multiplied by the output of a sigmoid function that decides which parts of the cell state are relevant [25, 29].

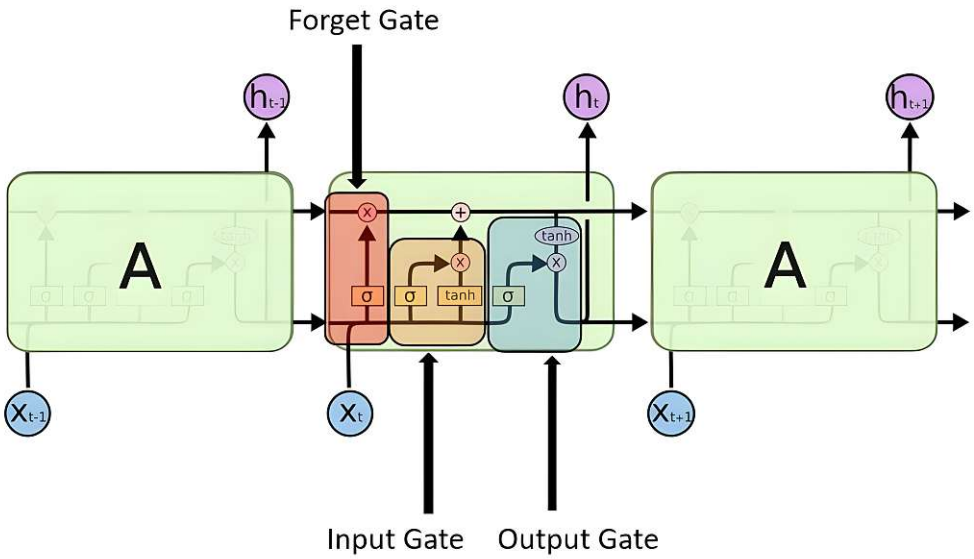


Figure 2.8: LSTM from [25] based on [29].

Nowadays, the LSTM is often more favorable compared to the basic RNN, as it shows remarkable results for a lot of learning problems. The learning strategy of this thesis is also considering LSTM to learn necessary features for the attention- and the agent-NN from the input sequence of video frames.

2.2 Learning Paradigms

Different DL-techniques can be categorized as follows:

- Unsupervised learning
- Supervised learning
- Reinforcement learning

Unsupervised techniques are based on a training process that does not use labels for the input data, accordingly, there are no target classes for the training data. Hence, this approach is interested in pattern recognition or finding higher-order correlations. Another possible application for the unsupervised or generative neural network is its usage as a preprocessing step for a supervised learning approach [21].

The *supervised* technique considers labeled training data, popular usages are classification tasks, where the learning approach is based on matching the correct class to a specific label image. The challenge of such classification applications is the fact that these training datasets have to be huge which makes the labeling a time-consuming task. Common architectures for supervised deep networks are CNN, MLP, and RNN which are continuously enhanced and improve the model optimization. The process for this supervised learning strategy has to contain certain steps:

1. Collect a dataset of expert demonstrations
2. Train a policy using supervised learning
3. Directly apply the learned policy in the target environment

The collected dataset has to be of good quality for representing labels that would fit the real world. Based on this data information, a NN can be trained and the results can be observed in simulation. Afterward, the model has to be deployed to the target environment and reviewed for reasonable results in this setting [23, 21]. This DL technique is essential for autonomous driving to take the necessary decisions. Supervised learning is based on a bunch of data and the aim is to learn and imitate a human driver's behavior [30].

Classical reinforcement learning is a reward-based approach with a learning agent that tries to maximize the extrinsic reward. The *reinforcement learning agent controller* (see Figure 2.9) acts as follows: The agent takes several discrete time steps within a specific environment, and for each of the time steps the agent makes an observation. Based on that observation the agent takes an action, furthermore, this action will cause a defined reward and aims to improve the model's actions. The definition of the rewards for all the agent's actions can be quite cumbersome [31, 32].

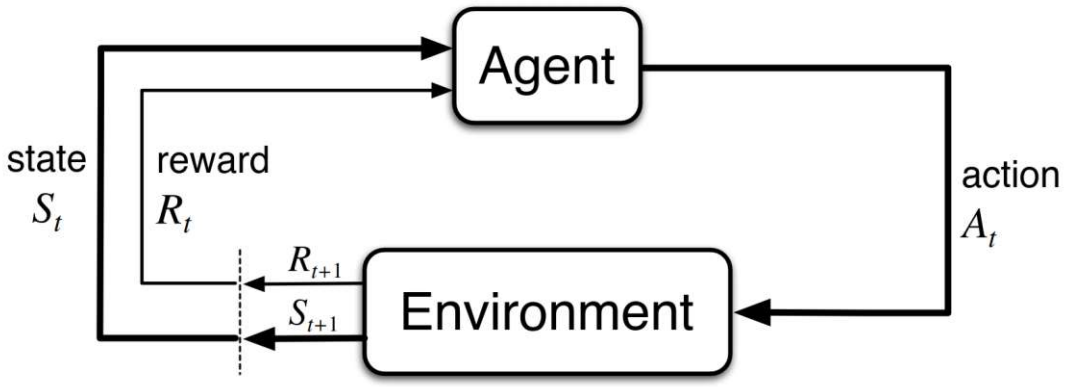


Figure 2.9: Reinforcement Learning (RL) agent controller from Sutton and Barto.

The classical ways of DL are competent for a lot of tasks, still, there are real-world applications for which these traditional ways of learning are not performing conceivably. In the best case, ML architectures have the availability of a huge bunch of labeled data where both the training, as well as the validation dataset, has the same distribution. The

gathering of this needed amount of data can be a difficult and time-consuming part of the work in machine learning. Hence, another way is *transfer learning* which is proficient to use DL on a smaller dataset. The focus of transfer learning is to transfer knowledge across domains to deal with the issue of a smaller dataset. The idea is that for specific learning tasks, generalization makes it easier to apply it to another similar task. For example, when someone knows to play harmonica, it will be easier to learn piano because there are synergies in the knowledge about playing a musical instrument. That means that for two learning exercises that have a relationship between them, transfer learning will be a good choice [33].

DL network architectures use existing layer structures as building blocks and integrate more features or layers to that structure to increase the complexity. This strategy is a core aspect of a neural network's benefit because it is possible to put together simple structures into more complex ones and learn more features. This circumstance makes it possible to use pre-trained models for feature extraction in an existing network, this form of network integration is known as transfer learning. The pre-trained models are available for download and e.g. Pytorch [17] can be used for integration into existing networks. Additionally, the usage-level is able to be adjusted considering the available volume of the dataset. This is done by the possibility to set fixed weights in the pre-trained model for specific layers and do the finetuning part only on the other layers. Specific features that are related to an individual dataset are more likely to be located in the training of the later layers, therefore pre-trained networks are used to transfer the ability to detect features of earlier layers and train the later layers based on the individual dataset. The architectures of existing pre-trained models have chosen complexity and the use cases of the applications are beyond classification tasks [27].

For the implementation of the agent-NN (see Chapter 5) the discussed approach of transfer learning was used. Therefore different pre-trained models [34] like *VGG* and *ResNet* were integrated into the encoding part of the model architecture, and the evaluation of the results is done in Section 6.2.

The design of *VGG* is inventive, as it increases the networks-depth by lowering the size of the used filters. The developments are done with 11-19 layers for the NN depth, where the models starting with 16 layers are superior to the models with less than 16 layers. The *ResNet* architecture was the first designed NN classifier that was able to perform results like a human. It starts with 18 layers up to a total of 152 layers. The resulting performance for big neural networks like the *ResNet152* is impressive, but on the other hand, some disadvantages arise with the increasing size of the network. The model training with 152 layers is cumbersome, the large numbers of operations necessary within the network change the size of the gradients, therefore the gradient flow between the layers may be slow or blocked. Another issue is the slow convergence of the learning process because the time needed for the convergence of neural networks with this size is often not really acceptable [35].

2.3 Loss Functions

The process flow of ML includes the generation of a predictor map where each of the predictions is a model's decision on which prediction fits the hypothesis of a specific problem the best. A loss function describes the loss value $L((x, y), h)$, where x is the input features, y is the corresponding label for the input, and $h(x)$ is the hypothesis for input features resulting in a predicted label outcome of the model. The size of the loss value has to be established by the difference between the predicted label $h(x)$ and the correct label y. A small loss value, therefore, states a small disparity between the predicted label and the correct label for a specific input data point. The overall aim of ML techniques is to find an adequate hypothesis that creates the minimum loss for any input data points. Depending on the application task the correct loss function has to be settled that fits best for the learning needs of the model [23].

The correct choice for an adequate loss function is essential for a proper training process and a resulting NN that can successfully work on the task it was created for. Depending on the target number a prediction has to consider two types of loss functions: *Binary*- and *Categorical*-targets. The *Binary* version aims to make a prediction $\hat{y}$ and uses a sigmoid activation function to scale the output to a number between 0 and 1, representing the probability e.g. a class got detected. In difference to that the *Categorical* loss function is oriented on multiple class predictions with k classes, this loss function is referred to as *cross-entropy* loss and is common for the usage of multiple classes. For real-valued outputs e.g. a float value representing a steering angle, it is typical to use regression learning with a Mean Squared Error (MSE) for the linear activation. Regarding the difficulty it can be stated, that the optimization for cross-entropy problems can be done more easily than for learning problems that have to consider a squared loss [27].

2.4 Regularization Techniques

Overfitting is one of the main challenges in ML, the issue occurs for a model that learns a hypothesis based on a training dataset and overfits that data. Hence, it performs well for the training part and shows a small loss, but unfortunately, the validation error is bigger on other data. The fact is that the model does not generalize well concerning the amount of training data it receives and this leads to the fact that the performance is not favorable for unseen data [23].

The probability for overfitting within a neural network is higher when the NN has a complex structure and the corresponding dataset used during the training process is rather small [27]. Therefore it is not always favorable to create the most complex models for specific tasks, especially when the data information quantity may not be sufficient for the model to be accurate for both the training- and the validation-dataset. The avoidance of overfitting can be reached by adequate regularization techniques.

Regularization is the most promising approach to avoid *overfitting* for the training process of a Neural Network (NN). Based on research experience with respect to the complexity of a model, it can be concluded that it is preferable to use a rather complex model with regularization than a simpler model without regularization. A categorization is possible regarding penalty-based regularization, where the most prevalent method is L_2-regularization. For L_2 the penalty is defined as the following regularization term: $\lambda * \sum_i w_i^2$, where $\lambda > 0$ and with different values the impact of the term can be adapted. The higher λ, the more strength the regularization on the sum of all squared model weights. The update within the NN has to be done for all weights by gradient descent. In comparison to that the L_1-regularization considers the absolute values for the weights of the network: $\lambda * \sum_i |w_i|$. Comparing both L_1- with L_2-regularization, L_2 is most often superior in respect to the accuracy and therefore more likely to be used compared to L_1 [27].

Another way to improve the generalization of a ML network is training with a bigger dataset. Unfortunately, this is often not possible or the necessary labor- and hardware-resources are still not available to increase the dataset. This leads to the circumstance of several ideas to fake data reasonably, one concept is the usage of *dataset augmentation* to enrich the dataset to a preferable training size. Especially for object recognition, this technique is advantageous because a high dataset diversity can be achieved by scaling or rotating existing images as transformation and adding the new variants to the existing dataset [36].

Dropout is a regularization technique that samples alongside all existing nodes in the NN and drops specific ones intending to create a model that does better generalize on a validation set. Output nodes cannot be dropped, because the dropout deletes the node including all incoming and outgoing connections and this would lead to prediction- and loss-calculation issues. From the input- and hidden-layer nodes are normally dropped with a probability between 20%-50%, the weights for the rest of the persisting nodes have to be updated after the dropout by back-propagation [27].

CHAPTER 3

Dataset Generation

The work of this master thesis is split up into the essential parts:

- ◇ Generation of a labeled dataset of attention frames
- ◇ Attention model for a human-attention prediction on a frame-based input (see Chapter 4)
- ◇ Agent model for a driving command prediction on a frame-based input (see Chapter 5)

The area marked blue in Figure 3.1 represents the input frame that will be forwarded to the attention-model to enrich the frame with the attention area. For learning the relationship between an input frame and a recorded human-attention, a corresponding dataset had to be generated in the first place. Therefore, to create this labeled dataset, a collaboration force was used to get high-quality human-attention frames for the training of the neural network models. This collaboration contained three students: *Felix Resch*, *Daniel Scheuchenstuhl*, and *Stefan Ulmer*.

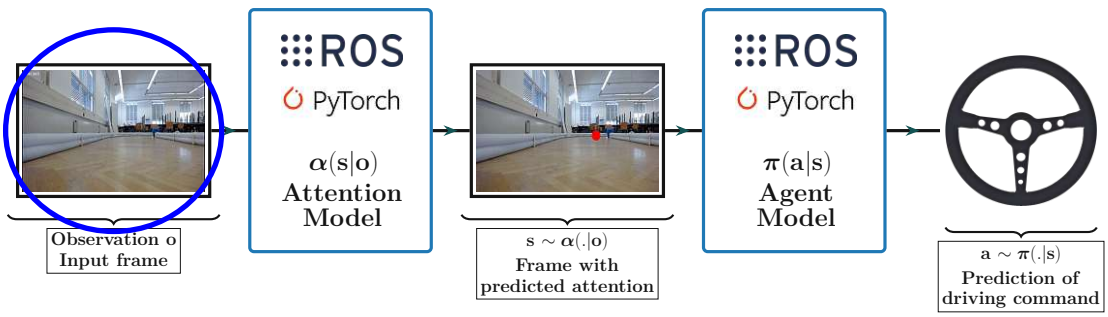


Figure 3.1: Attention-Agent prediction workflow (dataset-part marked blue).

The dataset was created on different tracks at the *Vienna University of Technology*. That leads to a preferable frame variation, a diversified dataset with adequate quality is an essential base for machine learning. On each track, multiple runs were made to create a dataset of the necessary size. The frames used during the training of the neural networks were split into a training- and validation dataset. Data augmentation techniques [37] (see Section 3.2.3) were used to extend this training dataset for more diversified training frames.

A part of a recorded run on the track is uploaded to a Git-repository [38] for demonstration purposes of how the human-attention is extracted and used as ground truth. The information for the attention focus of a human is used from the VPS glasses, which provide that information as a point with x and y coordinates. An input-label pair in the dataset represents a raw input frame (marked blue in Figure 3.1) with corresponding x and y coordinates of human-attention as ground truth. This information is used as input for the model training of the attention NN.

3.1 Eye-Tracking Technology

To gather a human's attention focus for the specific frames during driving, the work included the usage of eye-tracking glasses from a Viennese company named View Point System (see Figure 3.2).

(a) VPS logo. (b) Eye-tracking glasses.

Figure 3.2: Viewpointsystem eye-tracking glasses.

The assembled car including the Logitech camera is visible in Figure 1.3a. Both the eye-tracking system and the camera on the car were needed to create the dataset. The process of the dataset generation can be split up into several tasks:

- ◇ Record a video from the car's point of view with the mounted camera
- ◇ Record a video with the eye-tracking glasses for the same run
- ◇ Transform and synchronize to get a human's attention for a specific frame

In the first step, the human driver has to calibrate the eye-tracking glasses for the individual gaze. If the calibration of the VPS glasses is not good enough, there may exist an offset for the tracking of the human's pupils, and this lead to a bad or useless quality of the recorded human-attention. Only with good calibration, we were able to

retrieve the necessary information and use it for our generated dataset. Like playing a video game the human driver is focused on a screen wearing the VPS-glasses and using a controller for the driving commands. For the video stream on the screen the car's mounted camera is used as input and the car itself is driving on the track based on the driving commands. The recorded video via the VPS-glasses includes the human-attention from the gaze position on the screen where the human driver was looking at.

The quality of the dataset depends on the synchronization of the video frames. This is necessary as the human's attention is frame-based and the attention-coordinates have to be stored for the correct frame. The outcome of this projection of a single frame is shown in Figure 3.3. The top left image shows a single frame of the recorded video from the VPS system. The laptop in this image displays the streamed input from the Logitech camera, that is mounted on the car. Additionally, the bottom left is showing a recorded frame from the webcam directly stored on the car.

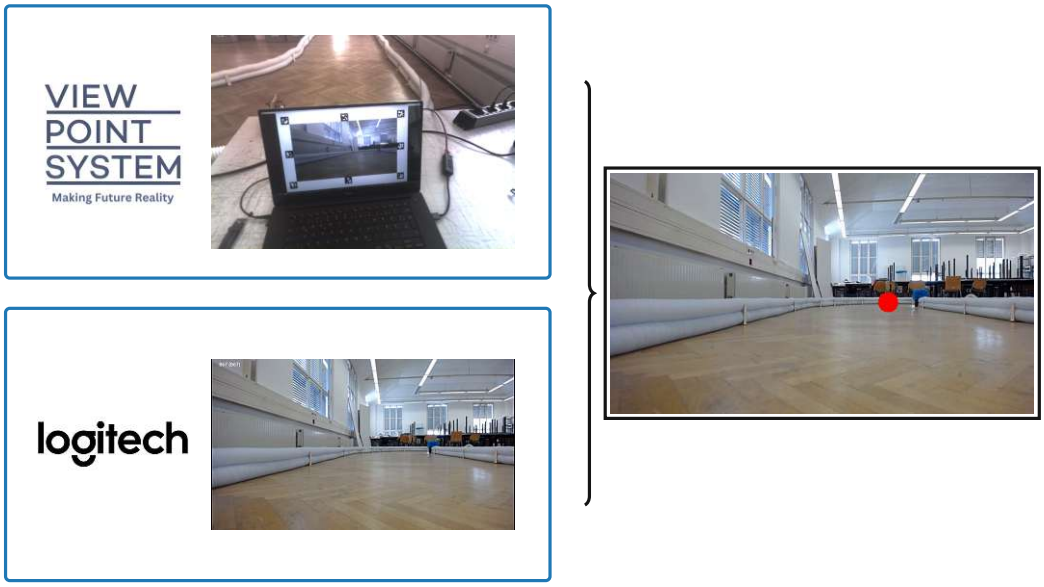


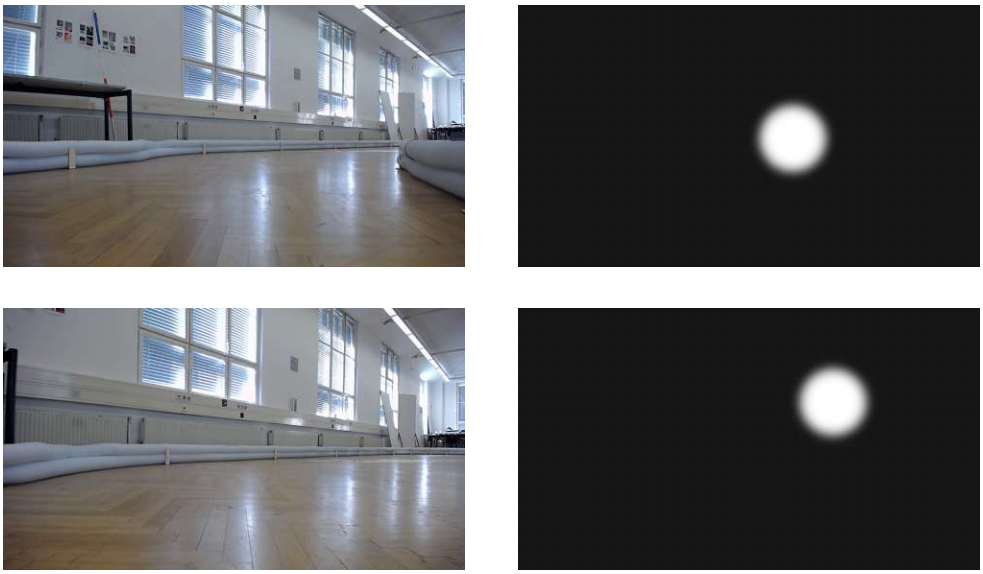
Figure 3.3: Attention projection to a single frame.

Because the VPS system records the video with all the surroundings nearby the display, a projection of the attention-focus-coordinates has to be done to the frame alone. This necessity is reasoned by the fact that the coordinates are based on the bigger VPS frame size and they are needed on the smaller size of the directly stored frame from the webcam. The result of the x and y coordinate projection is visible on the frame on the right side in Figure 3.3, and the transformed coordinates are marked as red dots on the raw input frame from the car camera.

3.2 Dataset

The setup including the eye-tracking glasses is used to generate a dataset that considers the human-attention areas of all frames. This information is necessary for the attention NN training, considering the x and y coordinates to learn the relationship between the human-attention area for each frame. Furthermore, additional data had to be recorded for the training of the agent NN because the final result has to be a steering angle to operate the car on the track (see Figure 5.1). Therefore, the driving command was recorded as steering angle via ROS-bags when the car was driving on the track. The F1/10 framework is able to consider the driving commands by incoming ROS-topics, these topics were recorded during run-time to gather useful extra information from the driving commands. The steering angle got synchronized for each frame like this is the case for the x and y coordinates of the human-attention.

The dataset is used to train a model to predict attention and a model for the prediction of the steering angle. For supervised learning, each frame needs a corresponding label. The neural network that should predict an attention area, uses a binary mask for the prediction of the human-attention which represents a specific range around the coordinates from the VPS eye-tracking system. This approach is visible in Figure 3.4, the input for the neural network are input frames like the examples shown in Figure 3.4a. The corresponding ground truth are the binary masks based on the human-attention x and y coordinates (visualized in Figure 3.4b) that are used for the comparison of the predicted output with the real binary mask label during the training process.

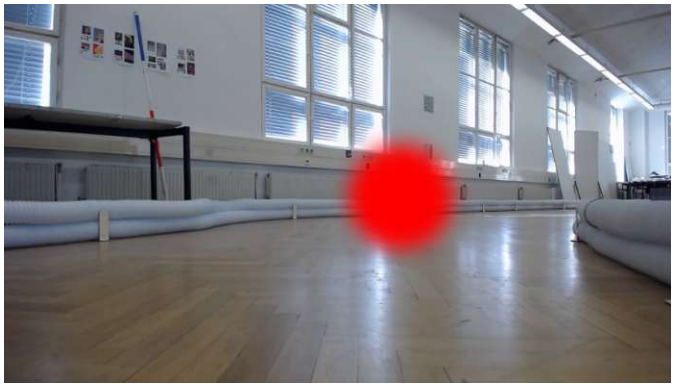


(a) Input frames.

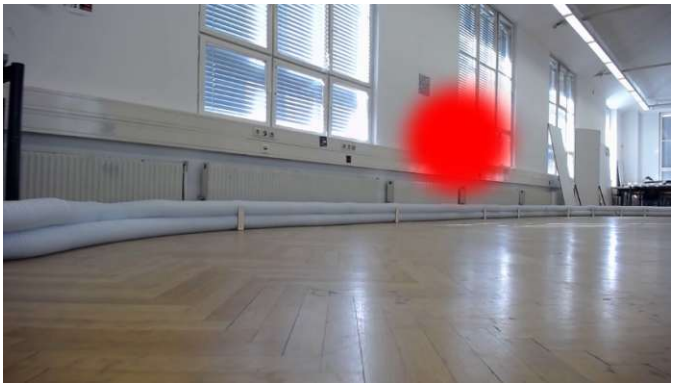
(b) Corresponding binary masks with circled area around the x and y coordinates.

Figure 3.4: Input frames and corresponding ground truth of the binary masks.

Accordingly, to the input-label approach of supervised learning, some examples of input frames used for the training of the agent model are shown in Figure 3.5. Each input frame has the human-attention marked as a red pixel area on it. From the recorded ROS-bags, each frame receives a corresponding angle label, representing the supervised learning input-pair of the agent model, necessary for learning the relationship of steering angles with the corresponding input frames. Two examples of steering angles (in radians) are shown for input frames in Figure 3.5a and 3.5b, a negative angle is representative for steering to the right. These steering angles will be predicted during the inference of the agent NN and the human-attention feature should have a supportive effect on the prediction accuracy.



(a) Input frame with a steering angle of [-0.11]



(b) Input frame with a steering angle of [-0.17]

Figure 3.5: Frame inputs with synchronized driving commands as a label.

3.2.1 Dataset Distribution

This section will discuss the dataset distribution of a specific track that has been used for the training of the agent and the attention model. The statistics shown are based on the recorded runs from the driving car in the lecture hall *Aufbaulabor* at the Vienna

University of Technology. A dataset distribution analysis will be shown for both parts of the dataset, the information about the attention coordinates as well as for the angle and speed information.

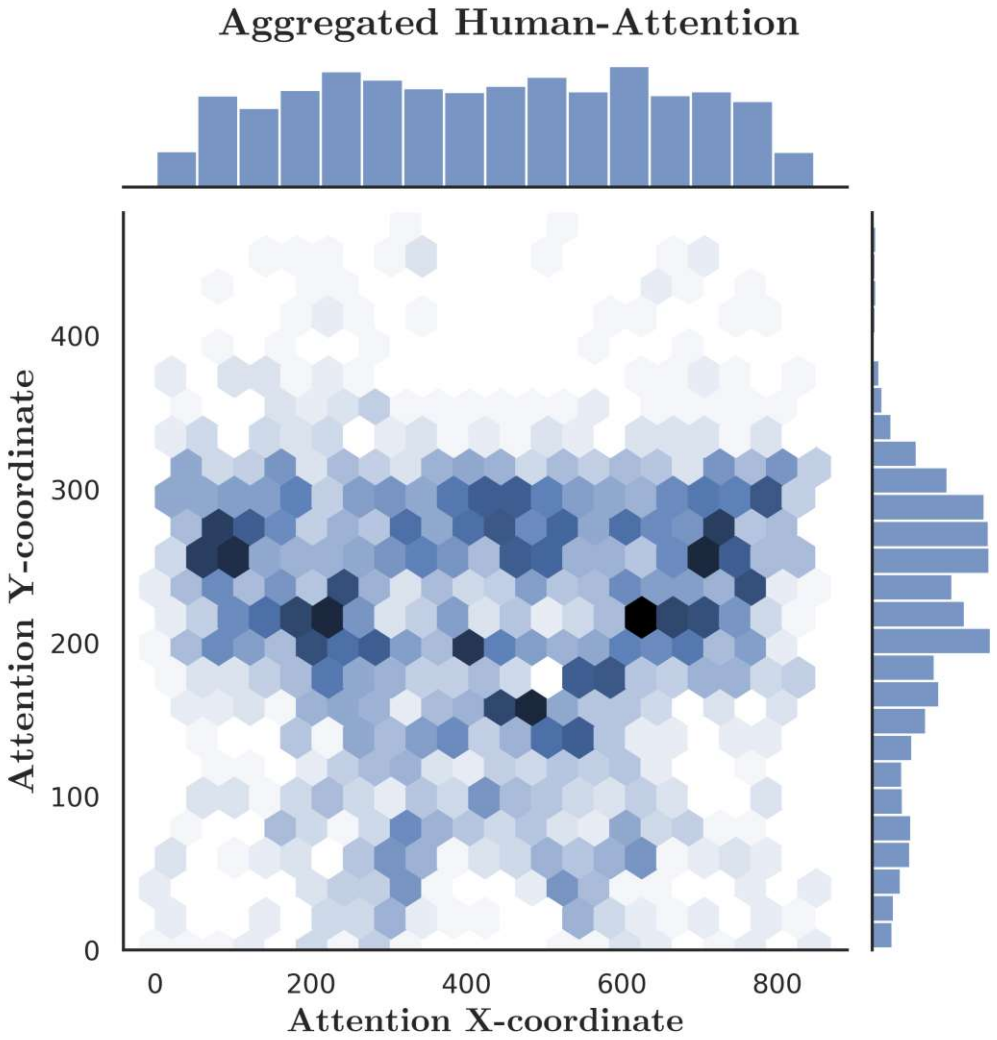


Figure 3.6: Hexagon plot for the frame-based attention distribution.

The information of the original attention coordinates is based on the frame size of 848x480, the distribution density of these coordinates is shown in Figure 3.6. Each of the hexagons represents a bin that unifies specific x and y coordinates. The position of the hexagons advertises the human-attention focus on a webcam frame. The darker the hexagon is colored, the more attention coordinates are placed in that bin.

A detailed distribution analysis is achievable when these attention coordinates are grouped regarding the frame width. A corresponding Kernel Distribution Estimation (KDE)

plot is visible in Figure 3.7. Therefore the attention coordinates are split up into two distributions regarding their x-coordinate. The coordinates that are smaller than half of the frame width are counted to the *left-sided* distribution and the others are considered to the *right-sided* attention gaze. For both the left- and right-sided attention there is an aggregation to the outer side which is shown by the increased intensity on the plot.

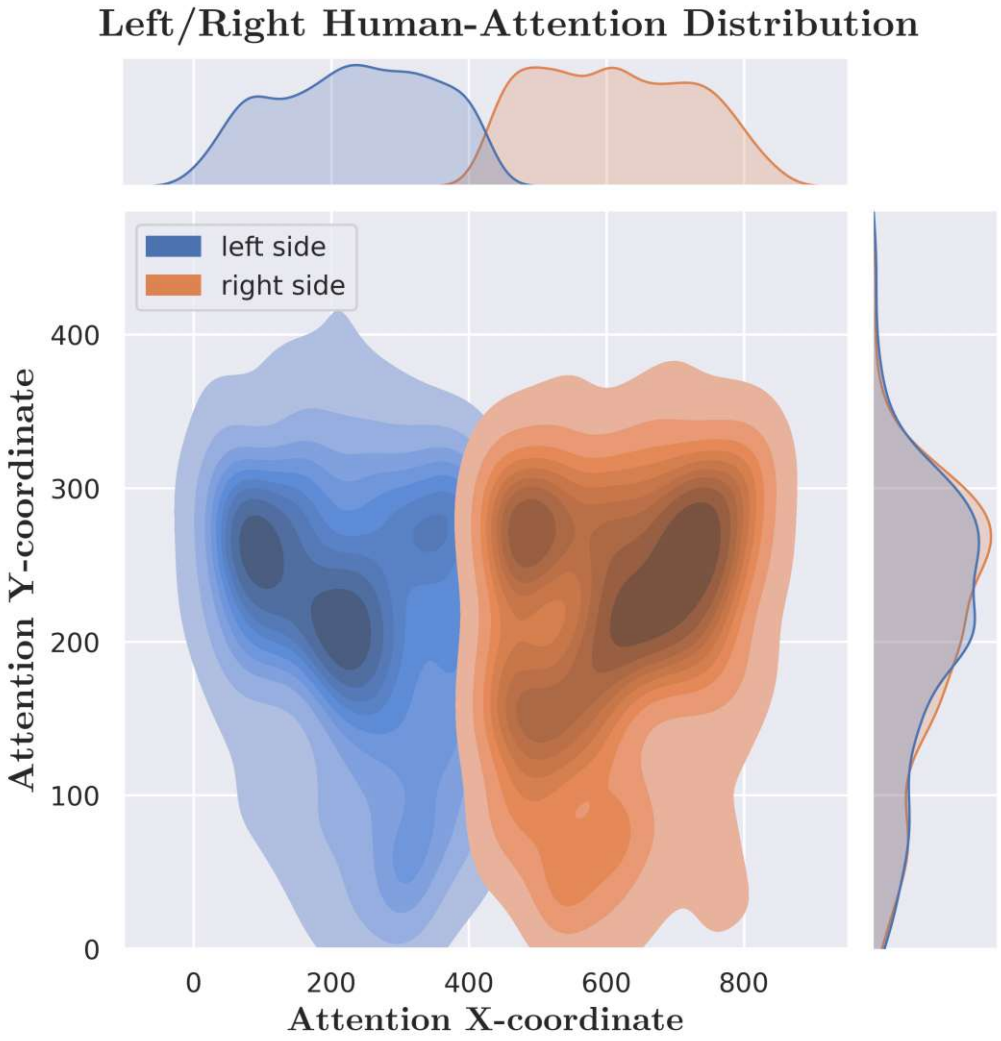


Figure 3.7: KDE plot for the frame-based attention distribution split into left- and right-side attention.

For the agent model, the dataset has been labeled with the synchronized steering angle from the user's input driving commands. For the dataset samples, each of these angles has a corresponding recorded video frame from the driving car on the track controlled by the controller input.

Regarding the sample distribution of the dataset, the allocation for all samples considering the angle is visible in Figure 3.8. The sample distribution shows a peek in the middle for angles ∼0.0, additionally, there are two smaller peaks around ±0.36. Based on the driving behavior this is an adequate distribution because for the majority of the frames, the car is driving on straights with low angles. When the user recognizes a curve on the camera stream he will try to steer whereas the steering angle is limited to the values about ±0.36 radians. For the model-based training process, the whole dataset samples were split up into a training- and a validation-dataset, where both provide a similar data distribution for a good representation of input & label pairs of the track.

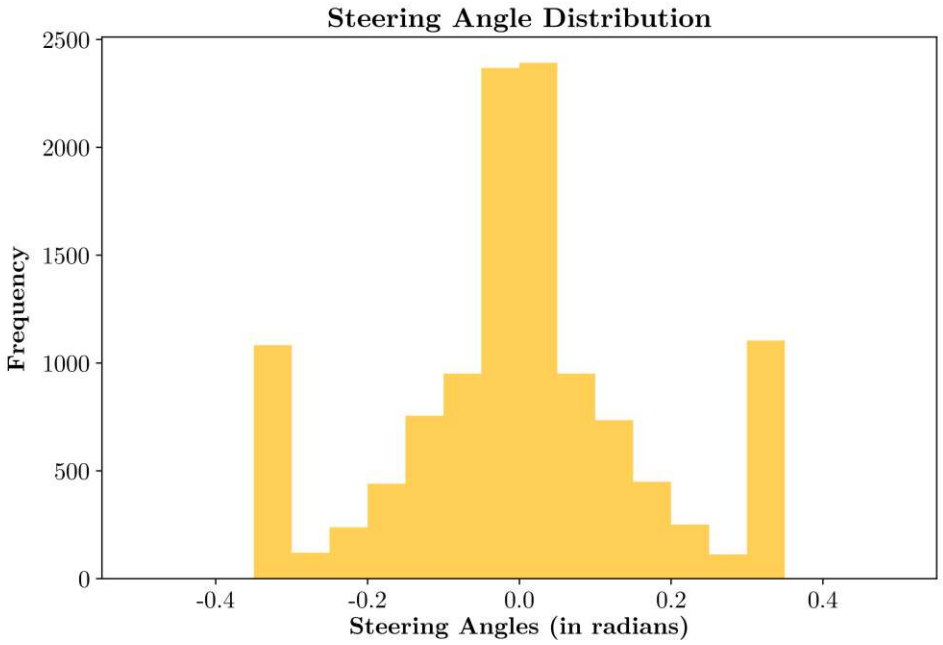


Figure 3.8: Angle distribution of dataset samples.

The distribution of the corresponding speed values of the frames from *Aufbaulabor* is shown in Figure 3.9. A clear peak arises at 1.5 m/s which was the limited maximum speed we set during the first recordings of the dataset. Another peak is at 1 m/s, as we did several runs for the same track and fixed different speed levels for the car to observe the outcome and receive better data distribution. With that setup, the input predominantly was the maximum speed we set for the running F1/10 system on the car when the user pushed the joystick in the forward direction. For other tracks, the controller was replaced by a steering wheel and pedals which allowed the user to reach a better distribution for the speed values during driving. Furthermore, we did start runs again from the beginning, when the driver was crashing because this would lower the quality of the dataset and we only want frames without uncommon driving commands as

this is the case for crashes. The speed was not directly used in the final training process of the agent NN, but the data could be used for future work discussed in Section 7.1.

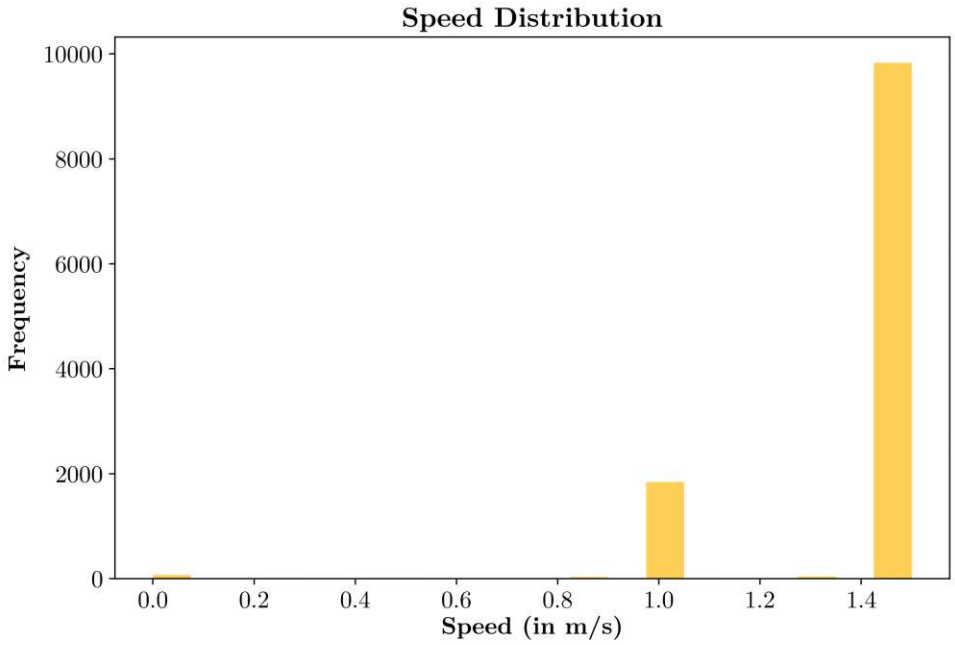


Figure 3.9: Overall speed distribution of dataset samples.

3.2.2 Dataset Sequences

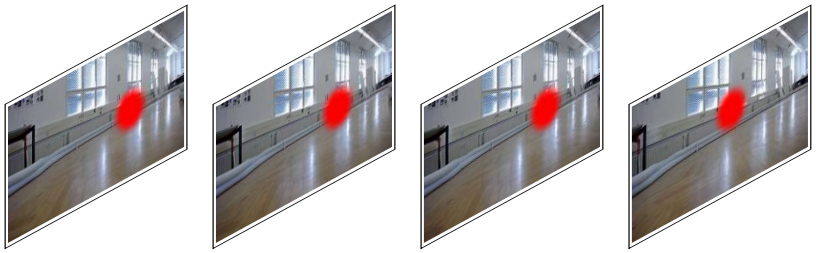
The generated dataset will be used for the model training with integrated LSTM layers. Generally, a LSTM layer is implemented for the use of sequenced data input, for detailed information see Section 2.1.3. The dataset is a frame-based representation with corresponding labels, therefore it was necessary to add additional logic for the training process to represent the frames with labels as a sequence. For single frame-based training, each input-label pair that is redirected to the neural network has one single frame as input and the label corresponds to that specific frame. In comparison to that for the sequenced version, each input consists of several frames and multiple corresponding labels.

Different numbers of frames were tested for adequate training performance because a huge increase in the training time was recognized with an increasing input frame sequence. Additional filter criteria were added for the sequence to provide a good trajectory for the frame sequence. The filter criteria also observe the quality of the attention focus alignment. In detail the filter discards sequences when one of the attention coordinates has an offset bigger than a certain threshold, ensuring that there are not too big *jumps*

in the steering angle or the attention area. Based on observations, when using 4 frames for each sequence, a lot more dataset sequences can be generated than this is the case for sequences that are bigger than 4 frames. The extra benefit is the improvement of the training process because the sequence of 4 frames needs less training and pre-processing time because the overhead is smaller for preparing the image sequence beforehand for a possible forward to the NN.

Examples of frame sequences are portrayed in Figure 3.10. They were used for training and had to be prepared as input for the inference of the NN models with LSTM layer. The attention-NN receives the sequence without attention area (see Figure 3.10a) and enriches a single output frame with an attention area. The sequence with attention in Figure 3.10b is necessary for the input of the agent-NN, the frame sequence enhances the information about the driving trajectory and improves the model's inference.

(a) Sequence of 4 frames



(b) Frame sequence of 4 images with human-attention.

Figure 3.10: Frame sequence for the attention- and agent-NN.

3.2.3 Dataset Augmentation

A common approach for increasing the dataset size is data augmentation [37]. Data augmentation has multiple benefits, in the first place the aim is to improve the prediction accuracy and the robustness of the neural network. For the NN-implementations the existing dataset was the starting point that had to be extended by dataset augmentation to improve the learning possibility of the models. Several image transformations were used on the base images and the newly created ones are added to the existing dataset. With this approach, the intention is to receive a trained model that generalizes better on

other input data. Additionally, with the expansion of the existing data, the effort is less compared to the initial dataset generation because the newly created data does not have to be labeled.

The implementation of the dataset augmentation was done with the *torchvision.transforms*-package from Pytorch [39]. This library offers a lot of possible data transformations that are aiming to improve the dataset. The transformations are *tensor*-based per default, to enrich the dataset with transformed data, a few common methods were used to increase the dataset size by this *data-augmentation*.

The frames of the dataset are stored locally as *torch tensors* with the marked attention area on them, this is the default transformation with *ToTensor()*. The specific input frames are loaded from the local storage and redirected to the neural network as input for the training process. After the input frame was loaded the transformations are performed to accomplish the data augmentation. Either the *GaussianBlur()* (see Figure 3.11b) is used for blurring the whole image or *ColorJitter()* for color transformations. The color transformations are divided into two separate variations: One augmentation considers the brightness & contrast (see Figure 3.11c) and the other one aims to adapt the saturation & hue (see Figure 3.11d). Detailed information about the transformations is available in the Pytorch documentation [39].

For all of the described ways of data augmentation, normalization was added as an additional transformation. Normalization improves the training speed and yields a faster convergence during the training of the neural network. Furthermore, the architecture of the agent neural network (see Chapter 5) is divided into the encoder- and decoder-part. The encoder uses different pre-trained Pytorch models, according to the specification the input for these models [40] has to fulfill certain properties. In detail the frame has to be resized to an image size of 224x224 and needs to be normalized with mean $\mu = [0.485, 0.456, 0.406]$ and standard deviation $\sigma = [0.229, 0.224, 0.225]$.

(a) Original input frame.

(c) Adapted brightness and contrast.

(b) Adapted with *GaussianBlur*.

(d) Adapted saturation and hue.

Figure 3.11: Input frames and corresponding data augmentation.

CHAPTER 4

Attention Neural Network

The generated dataset (discussed in Chapter 3) is the necessary base for the machine learning part of this thesis. The labeled dataset is the starting point, a ML approach has to be used to extract the information of the human's attention from the dataset frames. A neural network considering the attention information was designed for supervised learning and had to be trained with the specific input-label pairs of the dataset. The aim of the attention NN is the prediction of an adequate attention area for an incoming frame on the car.

Considering the whole workflow of this thesis, the attention NN part is settled directly after the step of the dataset generation and marked as red in Figure 4.1. When the implementation of the attention model is finished, an accurate prediction for the human-attention is necessary to enrich the input frame with this prediction. The predicted attention has to be marked as a red pixel area on the input frame and will be forwarded to the agent NN for the prediction of the steering angle.

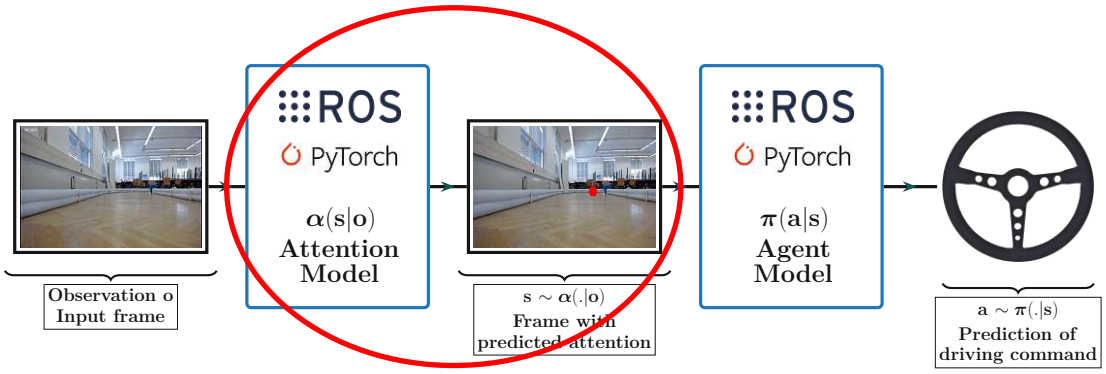


Figure 4.1: Attention-Agent prediction workflow (attention-part marked red).

This chapter contains the implementation that is fundamental for learning attention areas based on the labeled dataset. The attention area information from the dataset is considered as a segmented part of the image, a common approach to learning these segmented features is the usage of *U-Net* [41]. Accordingly, this U-Net architecture is used as the base for the work on the human-attention NN, and the corresponding layer structure is discussed in more detail. Furthermore, several adapted, improved, or extended versions of U-Net are characterized for the use of the dataset with the attention feature. The evaluation metric used for the accuracy analysis of the neural network is discussed in Section 4.2.4.

4.1 Attention Mechanism

The base for the learning process is the dataset including human-attention labels (based on attention coordinates from the VPS) which should guide the attention model to focus on these attention areas for other input frames.

The recent interest for *attention* in combination with machine learning, has also been studied in other topics like neuroscience and psychology. Attention seems like a conventional thing, but after numerous experiments to quantify and understand the process, there is still no clear concept for attention. Despite all that observations, attention is meaningful for information processing in the brain. Based on the VPS system that is used for the dataset, eye movements are indispensable because they occur several times per second and mark the x and y coordinates on the driving frames. When it comes to eye movement there is a tendency that image patterns like color contrast or intensity affect the attention attraction [42].

There are several possibilities for the integration of an attention mechanism into existing neural networks [43]. The attention mechanism can be established as a block and integrated like a plug-and-play system into existing neural networks. The objective of these methods is an improvement in the information-extraction part from the image features.

Most of the ways for integrating the attention mechanism for a neural network are based on the perspective of adding the mechanism as an extra feature for the specific network. These model implementations are often done for unsupervised models, therefore the input used for the training does not have a corresponding label that could guide the model. Therefore Wang and YANG proposes an implementation approach for supervised learning where the attention mechanism got implemented as a recurrent neural network into the neural network. They observed an improvement in the model's performance when the neural network considers supervised learning in comparison to the model that does not use supervised learning [44]. This supervised learning rudiment will be used for the training of the attention-NN, because the prepared dataset provides the needed input-label pairs as input frames with the corresponding x and y attention gaze coordinates.

This thesis idea is to learn the areas based on the different frames and therefore the approach has to consider feature extraction on a 2D image input. The research lead to an approach mentioned by Nvidia [45], generally, the implementation has to use image segmentation. The concept is segmenting the input frame into a simplified representation for easier investigation. The recommended architecture is called *U-Net*. The original design of this 2D-based image-based learning is mentioned from Ronneberger et al. and the proposed architecture is shown in Figure 4.2. The architecture of U-Net considers an encoder and a decoder part. The encoder part uses 2D convolution and max-pooling layers for downsampling the input frame. The decoder part follows the encoder and has to reverse the downsampling operations. This is done by upsampling the frame again considering the same parameters that were used for the downsampling process. Furthermore, this reference was used as the basis for the integration of a LSTM layer (see Section 4.2.3) to use and learn from frame-sequences for the training of the neural network.

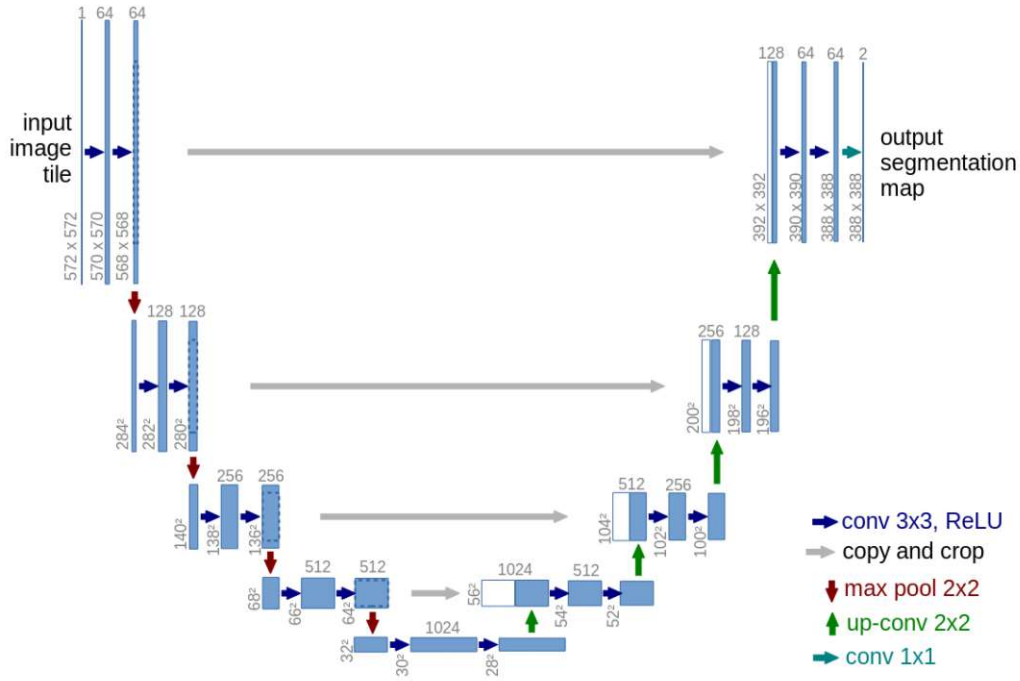


Figure 4.2: U-Net architecture of the original paper from Ronneberger et al..

4.2 Attention Model Implementation

The implementation of the attention-NN is the first part of the neural networks that had to be designed for the prediction process workflow in Figure 4.1. The corresponding repository of the attention model implementation is available in [46]. The agent model (discussed in Chapter 5) is trained with both the Human Attention Frame and the frames without attention for a comparison of the improvement including the attention areas. Therefore the attention prediction from the attention NN has a big impact on the overall prediction outcome of the driving commands from the whole prediction process. The attention-NN is responsible for the attention prediction and the predicted binary mask is marked on the input frame and redirected to the agent model. If the predictions of the attention areas are not reliable, the agent model that considers these Human Attention Frames will not be able to predict the best possible driving commands. This shows the necessity of an attention NN that is well-trained for segmentation features and specifically for predicting a human's attention.

The implementation based on U-Net as a reference is emphasized in Figure 4.3. This implementation considers a single frame input and the described encoding and decoding part.

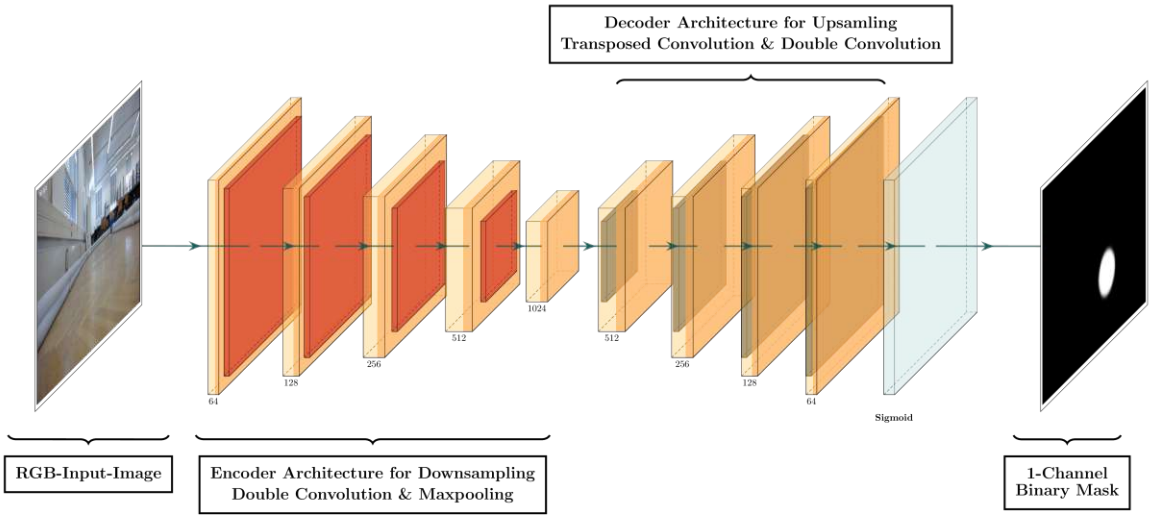


Figure 4.3: Adapted U-Net architecture.

Attention is an interesting topic, especially when it comes to the correctness check. The recorded dataset includes a human's attention information for different tracks, often the attention areas are *jumping* from left to right because the eye movements of the human driver happened in the same way. This makes it quite challenging to always state what exactly is the correct attention for a specific frame. Additionally, the attention topic has been analyzed by scientists, but they are still not aligned and therefore it is not clear

how to exactly specify attention [42]. The metrics used for measuring the accuracy of the attention NN predictions are discussed in Section 4.2.4. A detailed evaluation and comparison of the different implementations of the attention NN are done in Section 6.1.

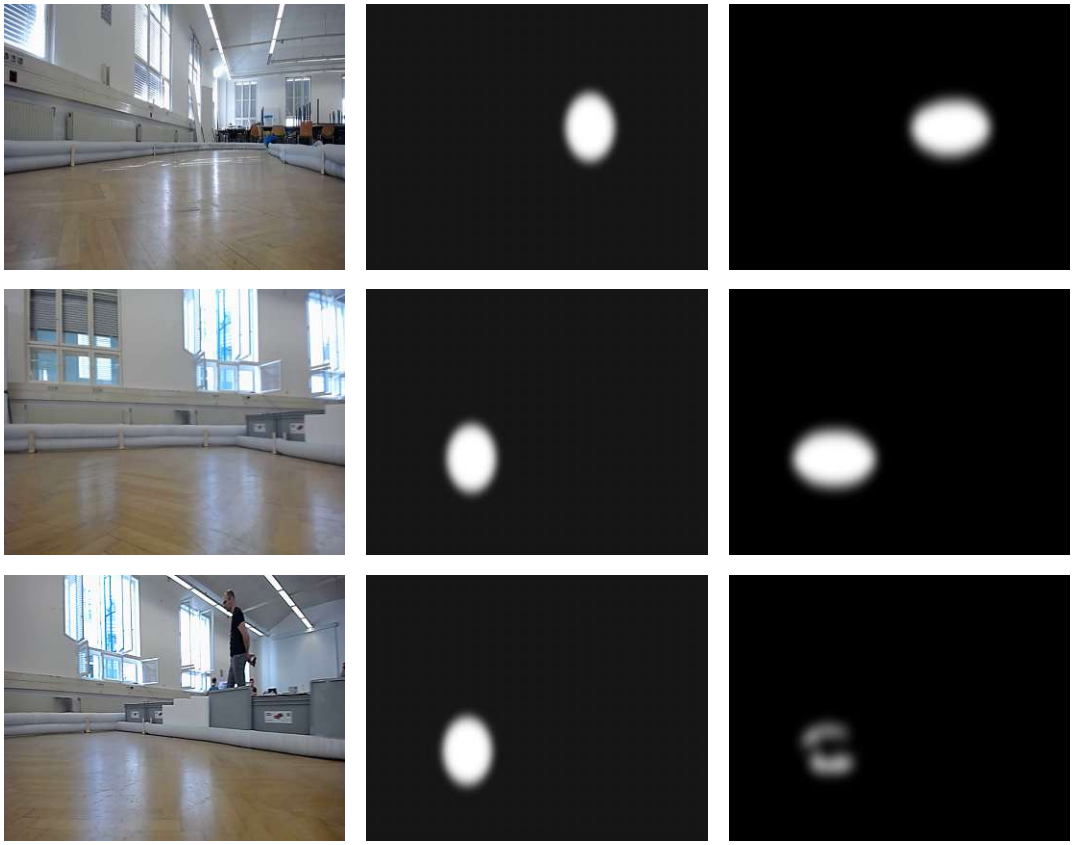
4.2.1 Attention Prediction

Considering the attention mechanism and the U-Net architecture as a baseline, the extracted feature information is based on 2D input frames. The layer structure of the U-Net architecture considers 2D convolutions to extract features from the 2D input frames. After the upsampling tasks in the decoding structure of the U-Net structure, the outcome is again a 2D image with the same image size as the input. The 2D output of the neural network was set to 1 channel, this means that the model predictions are binary-based. An example of this 2D binary mask can be seen in Figure 3.4.

The neural network, therefore, provides a pixel-based probability where the attention area is located on a specific input frame. The binary mask is mapped as an attention area with red pixels on the input frame. This Human Attention Frame is the base for the prediction of the driving commands because this frame with the marked attention area is redirected as input to the agent neural network.

A few prediction examples of the binary mask for an input frame of the *Aufbaulabor* track are shown in Figure 4.4. The input frames are directed to the attention-NN and these frames are from the recorded data with the ground truth human-attention. The ground truth is displayed as a binary mask in Figure 4.4b, and the predicted attention from the neural network is visible in Figure 4.4c. The shapes for the predictions are not always circles like the ground truth, they are often drawn-out, and sometimes the predicted binary mask is not filled entirely or split up into smaller areas. This is the case when the trained model is not confident enough about the prediction and the probability to place filled pixels is too low.

The accuracy of a neural network prediction will be checked by the difference between the attention area that got predicted and the attention area on the 2D binary mask label. Possible metrics for stating the accuracy of a neural network based on 2D image features are shown in Section 4.2.4



(a) Input frames. (b) Ground truth binary masks. (c) Predicted binary masks.

Figure 4.4: Input frames with the corresponding label and predicted binary masks.

4.2.2 Attention Neural Network Architecture

Analysing the U-Net architecture on a layer-base it is recognizable that the structure is generally symmetric. As shown in Figure 4.2 the neural network starts with the 2D image input and downsamples the image in multiple steps. The model is learning the segmented features from the input. Corresponding to the downsample part, the upsampling is done with the same depth as the downsampling for the reconstruction of the original image size. The downsampling- and upsampling-part consist of blocks that are built with different layers, the detailed structure is illustrated in Figure 4.3. The encoding architecture is recognized as the *Downsampling*, where each of the downsample-blocks of the NN takes the image and puts it through two convolution layers followed by a max-pool layer to get features from the image. For the reversal of these operations to reach the original size Figure 4.3 considers the corresponding decoding part of the NN where each block in the *Upsampling* contains a Conv2dTranspose followed by two convolution layers.

The *U-Net*-architecture presented in Figure 4.5 addresses the attention-NN implemen-

tation with 256 channels and includes a LSTM layer. The default U-Net architecture stated by Ronneberger et al. considers 1024 channels (see Figure 4.3). The variations of the U-Net implementations including a LSTM are discussed and evaluated in Chapter 6, the final version that was deployed on the car considered the LSTM and 256 channels.

4.2.3 LSTM Integration

The model structure explained in Section 4.2.2 works on a single frame base with shuffled data which means dependencies between the different frames are not considered. For a driving scenario with ongoing movement on a track, it is preferable to consider these dependencies for a better model prediction over time. Therefore this section discusses the integration of a LSTM layer [47] that is capable to learn on a frame sequence considering historic states for current predictions.

The structure is advertised in Figure 4.5. The implementation loops over the sequence of frames and each frame is redirected to the LSTM after the encoding part of the U-Net model (for a detailed description about the functionality of LSTM see Section 2.1.3).

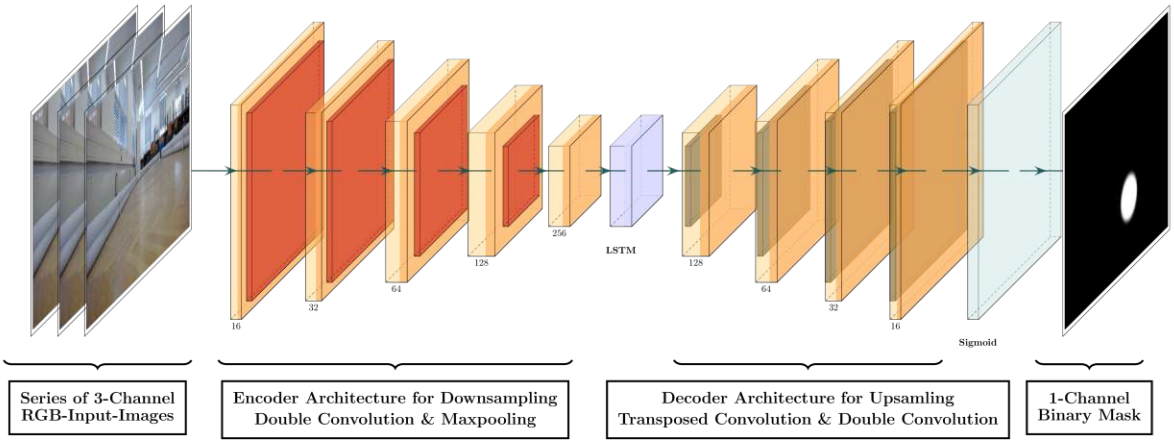


Figure 4.5: U-Net architecture with an integrated LSTM layer.

4.2.4 Evaluation Metrics

Respecting the evaluation of the implemented attention NN the quality of the attention area prediction will be measured by the accuracy and the loss during training and validation. The accuracy can be examined by the comparison of both the prediction and the label 2D binary mask. Each of the masks has a shape of 128x128 (like the input image size) with 1 channel where the single pixels are set to 1 if this pixel is part of the attention area. The training of the U-Net architecture is focused on the segmented areas

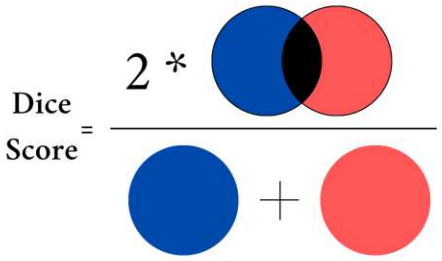
of an image, therefore the check is based on the comparison of these segments for the prediction and the comparable ground truth. The current science of attention is not fully explored and understood [42], it is still quite hard to state if the accuracy check based on this segmentation alignment is a reasonable approach. From the technical correctness concerning supervised machine learning this is the way to go, the label from the recorded dataset is the evidence for the prediction. From a semantic point of view, attention cannot always be classified that easily.

Nevertheless, the basis of the training process of the attention NN is the segmented features. The evidence check between the prediction and the label for an input frame is done during the model training with the specific loss functions. Accordingly, proper loss functions that consider the segmentation features and state the accuracy of a model's prediction are the *dice loss* [48] or *Intersection over Union (IoU)* [49].

The starting point for the *dice loss* is the dice score shown in Figure 4.2. The numerator considers the intersection between the prediction and the label binary mask, hence this is the sum of all pixels that are 1 in each of the masks. The denominator contains the sum of both amounts, the division result is a ratio between 0-1 that tells how many of the pixels are intersected and found in both masks. When the ratio would be 1, the segmentation is a 100 % overlap. Hence, the numerator has to be multiplied by 2 for normalization to reach a possible dice score of 100 % if the overlap of both binary masks would be a perfect intersection with all pixels of the attention areas set to 1 in both masks [48].

$$DS = \frac{2 * |A \cap B|}{|A| + |B|} \qquad (4.1)$$
$$DL = 1 - DS$$

4.1: Dice Loss Formula.



4.2: Dice Score.

The formula for the *dice loss* is described in 4.1. The formula variables are:

- A = Prediction binary mask in 2D
- B = Label binary mask in 2D
- DS = Dice Score
- DL = Dice Loss

The dice loss is the ratio of the pixels that are not 1 for both of the binary masks. Therefore the dice score has to be subtracted from 100 % to receive the remaining ratio for the dice loss.

The second classification number mentioned was IoU, also called *Jaccard* Index, which is similar to the discussed dice score. The calculation would be done in a similar way and the resulting ratio would range between 0-1 representing the accuracy [49]. Due to this similarity, for the training- and validation-process of the attention-NN, only the dice loss was considered for the following work of this thesis, especially for the evaluation in Chapter 6.

4.2.5 Finetuning

The training part of machine learning, especially for working out the best training parameters can be quite cumbersome. This is a common issue in the machine learning area and there are already existing tools that are used to find the best hyperparameters for the training process. The tool used for this thesis to assist the hyperparameter determination was *optuna* [18]. With this tool, it is possible to specify an *objective* that executes the training process including the training- and testing-loop for a neural network. Different parameter ranges can be specified for the optuna study, e.g. the learning rate ranging from 1e-09 to 1e-01 or the weight decay. During execution, different trials are done with the specified parameter ranges and the validation accuracy is tracked for analyzing which parameters achieve the best trial.

Furthermore, the best trial parameters from optuna are used as starting point for the training process of the neural networks. Additionally, an essential objective is the convergence of the training and validation accuracy of a neural network. The comparison of both accuracies often results in over- or under-fitting, this circumstance would be visible in a constellation where the training accuracy is too much above or below the validation accuracy and the oscillation would begin after a longer period of matching accuracies.

Based on the generated dataset (see Chapter 3) the trained attention NN tended to overfit on the training data. Different techniques were used to improve the implemented U-Net architecture of the attention neural network. The first consideration regarding the overfit on the training data was the dataset itself because the first training processes did only use the runs of a single track. With the extension of the dataset by adding multiple runs of other tracks, there was already an improvement regarding this overfitting issue possible. Another approach to deal with overfitting is the *reduction of the complexity* of a NN. The U-Net architecture as shown in Figure 4.2 considers an increasing number of channels for each of the blocks of the encoder and the decoder. For complexity reduction, the number of channels for each of the blocks was reduced. Other possible methods that treat the overfitting were the usage of *dropout* and *weight decay*. The dropout discards features during the training process which increases the robustness of the neural network. Both of these regularization methods are especially important when the amount of the used dataset is manageable [27].

CHAPTER 5

Agent Neural Network

The agent model is the second neural network of this thesis work and for the model training the same dataset was used as for the attention model (see Section 4). The dataset has to be a bit changed in the fact that for the agent model, the input frame will include the marked attention area compared to the raw input frame for the attention model. The agent was trained on a specific *steering angle* as ground truth for each frame. The steering angle has been synced to each frame for learning a human's driving behavior. This circumstance is essential because the prediction outcome of the agent model has to lead to driving behavior on the track and therefore the prediction accuracy has to be high to reach adequate performance on the real hardware F1/10 car.

Considering the workflow idea addressed in Figure 5.1, the agent model is the last part that is missing to predict the necessary driving commands for the F1/10 systems. When the agent-NN reaches a reasonable prediction accuracy for the steering angle, the whole processing workflow can be put together to predict a driving command based on a single input frame from the camera.

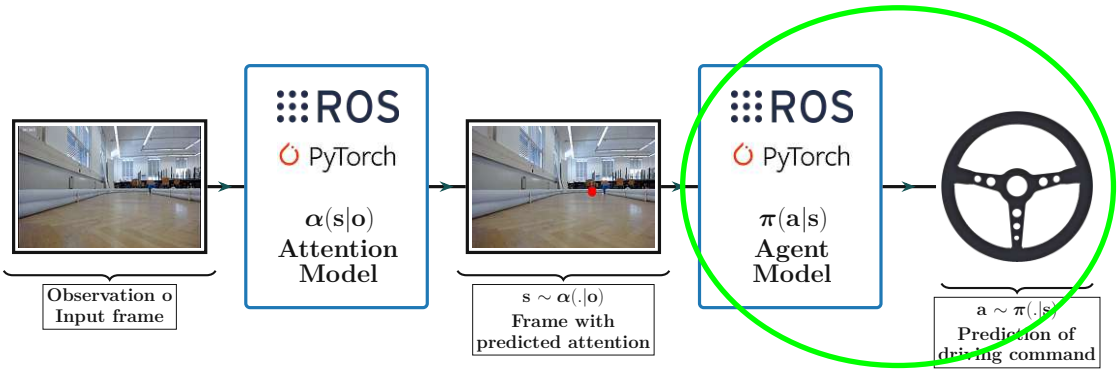


Figure 5.1: Attention-Agent prediction workflow (agent-part marked green).

Starting from the training with independent frames there was done a LSTM integration to observe possible improvements for the model prediction if frame sequences are used in a sequential order compared to independent frame training. An improvement for models that integrate LSTM in the layer structure [50] is based on the fact that a sequence of independent frames is not capable of learning a relationship between sequential image frames. In comparison to that with LSTM integration temporal information between the image sequences is recognized and used to improve a model's ongoing predictions. Common approaches working with machine learning are showing an improvement in learning techniques with LSTM integration for driving scenarios [50]. More on this and a general analysis of the agent model will be discussed in Section 6.2.

5.1 Decision Making

The final agent model has to be able to predict adequate driving commands to navigate the real F1/10 car on a F1/10 track. For autonomous driving to reach the state of self-driving, it is necessary to have proper decision-making for different upcoming scenarios.

One option to get autonomy could be the implementation of rules for driving behavior, these rules have to be encoded for maneuver selection and path planning. Multiple criteria can be used to make a decision based on a specific driving state: Sensor measurements, object detection, and traffic rules. In combination with path planning to consider obstacle movements in the environment, the idea is to decide on a driving trajectory within a reasonable time. The disadvantage of implementing these rules is the intense workload for engineers to figure out which scenarios are the most important for driving decisions. With increasing data and all the different scenarios at some point when the complexity is getting too far, this is no longer a feasible approach to be capable of adequate driving decisions. Another method that is able to solve this issue is imitation-based, which addresses supervised learning techniques to learn an agent policy by mimicking the driver's behavior. With that approach, it is possible to get rid of the manual implementation of all the rules for the sophisticated traffic environment. The focus on the driving behavior instead of the implementation contemplating all possible upcoming driving events leads to a better performance. The trained models are able to do lane-keeping on roads with and even without lane markings. A deficiency is the necessity for labeling the training data that is used for the supervised learning approach. The results of these implementations show that supervised learning is superior to manual rule implementation. Accordingly, supervised learning is a common technique for learning a driver's behavior based on a bunch of data with diversified driving scenarios [30].

Autonomous vehicles are computing systems that have to be qualified for decision-making in driving scenarios, often they use additional observations in the form of sensors or cameras. Some usual hardware resources used to gather this information are LiDAR and Inertial Measurement Unit (IMU), whereas a LiDAR was used on the F1/10 car to get the distance information to the border and other obstacles on the track. With

that systems direct usage of the sensor information is possible to control an autonomous vehicle [3]. The system architecture for the decision-making of an autonomous car can be split up into different blocks, together forming a pipeline of the combination of both deep learning and non-learning techniques. The work of this thesis mainly covers the usage of supervised learning techniques.

The input for the agent model is a Human Attention Frame and a corresponding steering angle representing the ground truth. Hence, the steering angle is the necessary driving command that has to be predicted for the real F1/10 car to drive on the track. Therefore, in addition to the video recording for the dataset, ROS bags were recorded to save the corresponding commands during driving. The commands of the ROS bags were synchronized to the corresponding frames of the videos and the agent-NN had to be trained on these pairs to learn the relationship.

The agent model has to predict numerical values, therefore this task is a regression problem instead of a classification problem. CNNs are known for their suitable performance of learning image features, this is a sufficient solution for the encoder part of the agent model that has to be capable of handling the 2D video frame input [51]. The CNN extract the features, followed by multiple linear layers to increase the complexity and improve the accuracy/loss of the model (details discussed in Section 5.3).

Furthermore, LSTM integration (see Section 5.3.2) mentioned by Jonah is an advanced type of RNN that is proficient to take the history of previous image features into consideration. Therefore it uses frame sequences for the training input to learn dependencies and relationships between the frames to make driving command predictions more reliable [51].

5.2 Command Prediction

The aim of the autonomous agent NN is a good prediction of driving commands based on the redirected Human Attention Frame from the attention NN. The prediction of the human-attention is realized by a binary mask and the filled pixels of the mask are signed as red pixels on the input frame for attention enrichment. The driving command predicted for the incoming frame is a steering angle that has to be published via a ROS-topic to operate the real F1/10 car. The quality of the steering angle prediction depends on the quality of the beforehand attention area prediction. If the attention prediction is reliable and therefore focuses on a specific area on the track, the agent NN can use this area as lead and improve the driving commands on that base. Figure 5.2 shows a comparison of predicted and label angles over multiple frames. Beginning from the left both values fit to each other whereas the last frame on the right advertises a tendency to the right for the predicted angle in the right curve.

Section 5.3 gives details about the implementation of the encoding and decoding part to reach the necessary output value of the steering angle prediction that represents the

Figure 5.2: Comparison of predicted (red) and label angles over multiple consecutive frames.

driving command. The general implementation considers the single frame input-based prediction. Hence, a non-reliable prediction of the attention area for a specific frame could result in a bouncing behavior of the driving command prediction. To avoid this behavior a LSTM layer has been integrated into the existing structure (see Section 5.3.2) for the handling of outliers.

5.3 Agent Model Implementation

The implementation of the agent NN is the second part of the prediction process that is advertised in Figure 5.1. The corresponding repository of the agent model implementation is available in [52]. A simple reference for an agent that predicts steering commands is proposed by Bojarski et al.. This architecture expects a 3-channel-RGB image as input for multiple convolution layers representing the CNN part of the model for the image feature extraction. After that, the features are flattened before they are redirected to the decoding part that is implemented as a stack of linear layers. The resulting single value output after the linear layers represents a car's steering angle.

Considering the idea of this architecture, this was the starting point for the agent NN of this thesis. The input frames of the generated dataset (see Chapter 3) are 2D 3-channel-RGB-based images. Hence, convolution layers are considered for the feature extraction, instead of stacking multiple of these layers to form the encoding part as CNN, the whole encoding part was replaced with an already trained model.

Pytorch offers different pre-trained models, that are common for feature extraction from 2D images, e.g. (*VGG* and *ResNet*) [34, 35], and they will be used for the encoding part of the agent-NN. Additionally, the output features of these pre-trained models had to be adapted (reworked model *classifier*) to fit the input shape of the following decoding part. The decoding part was implemented with multiple linear layers resulting in a single output value on the last linear layer for predicting the steering angle. Moreover, compared to the reference architecture, regularization techniques like *dropout* and *weight decay* were used to improve the robustness of the model.

5.3.1 Agent Neural Network Architecture

Starting from the Nvidia reference model [53] it occurred that there were too less features considered in the model architecture, therefore during training the reached inference accuracy was moderate for the prediction of the driving commands.

Due to this result, the model's complexity has to be increased to achieve better accuracy performance for the agent NN. The base architecture for the agent model considering pre-trained models for the encoding part to increase the complexity is shown in Figure 5.3. The training input is a dataset of input-label pairs of a Human Attention Frame with a corresponding steering angle as a label. The CNN-architecture of the Nvidia reference [53] is replaced with *VGG* and *ResNet* for the encoding part of the model. An integrated pre-trained model for an existing neural network architecture is called a *backbone* model, used to improve a current structure. Furthermore, the features will be flattened by the linear layers of the decoding structure to get the driving command prediction. The resulting steering angle for the output is represented as a float value and can be directly published via a ROS-topic to operate the car.

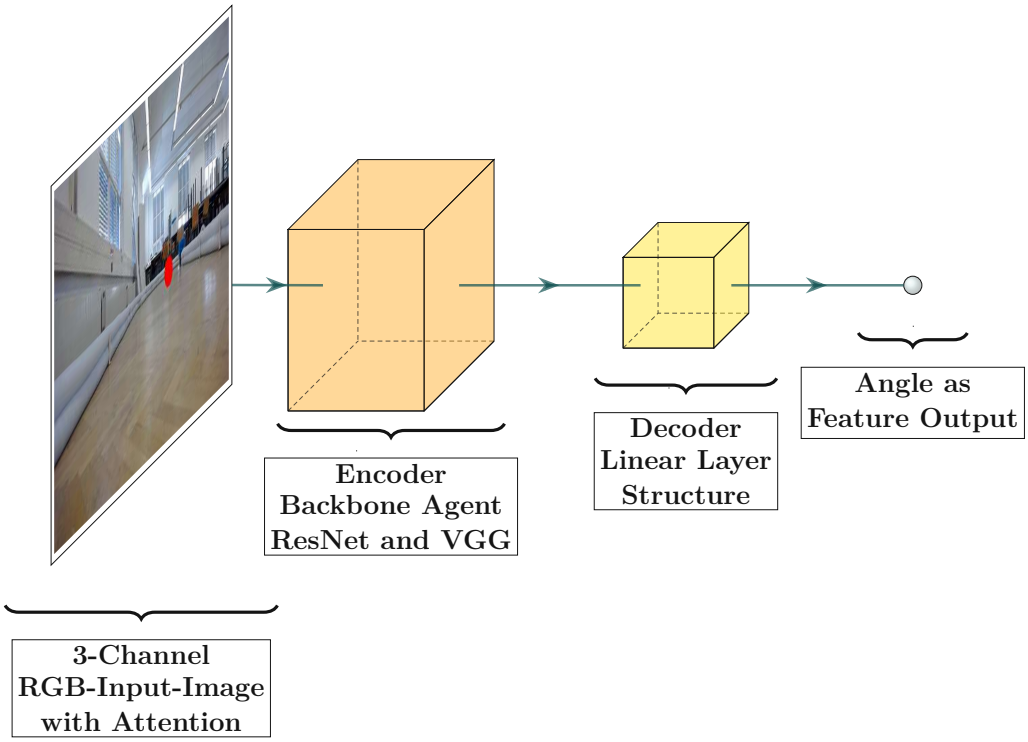


Figure 5.3: Overview of agent model architecture.

In more detail, the structure consists of an *encoder* and a *decoder* part. The encoder part is used for the extraction of features from the input frame, this happens by using different versions of common pre-trained models. The feature extraction is done on a 2D

input image base and the architecture therefore is composed of followed convolutional- and max-pooling layers. Additional features can be added by choosing more complex layer architectures for the CNN part of the model. An approach is *transfer learning* with an existing model that is loaded and used for feature extraction. The paper from Zhang. and Ohya. [50] describes the usage of *ResNet* for extracting the features from the images, for the agent-NN both *ResNet* and *VGG* are considered as pre-trained *Pytorch* models.

In comparison to the encoding part, the decoder is the *Fully Connected (FC) Layer* block (see Figure 5.3) of the agent neural network consists of multiple linear layers. Therefore the 2D input shape from the frames that are redirected from the 2D convolutions has to be reshaped so that the input fits the FC layers. Hence, the input is flattened before being redirected to the first linear layer. The architecture of the decoder starts with the number of input features based on the parameter size of the pre-trained model. The matrix multiplication within the layers is done until the output shape of the last layer is achieved and only a single value is the output of the neural network.

The implementation of the agent NN considered the mentioned pre-trained models as CNN architecture for the encoder part. The analysis on the usage of VGG and ResNet instead of the previous CNN encoder of the neural network is discussed in Section 6.2. Generally, the VGG16 as a backboned model reached the highest accuracy for single input frames with ∼86% (see table 6.1). The overall best accuracy for the agent-NN was reached with the integration of a LSTM explained in the next section, the additional gain by the usage of a frame sequence results in prediction accuracy of ∼93% when using the *ResNet50* as backboned model and integrating the LSTM to the agent-NN. The resulting neural network architecture is the base for the ongoing integration of the LSTM-part of the model. With that extension, the overall accuracy/loss should still further improve and make the model even more robust for different input frames when predicting.

5.3.2 LSTM Integration

The agent model shown in Section 5.3.1 expects a single input frame and the driving commands are predicted on the current state with consideration of this frame. The integration of a Pytorch LSTM layer [47] was done to consider multiple input frames as the sequence for the agent to improve the predictions. The newest incoming frame from the webcam will be added to the sequence of the previous frame sequence and the oldest will be dropped, forwarding the whole series as input for the model. The idea is that an agent model trained on sequences can avoid bouncing prediction values because it does not only consider the current state but also the input frames that happened beforehand (see Section 2.1.3 for the functionality of LSTM).

The abstracted version of the agent NN with LSTM is advertised in Figure 5.4. Generally, the LSTM layer from the Pytorch framework provides options to use the implementation with a for loop or redirect a complete sequence to the layer. With the change from the single-frame-based approach to a sequence of frames multiple updates were necessary for the agent model and the training process:

- ◇ Updating dataset input
- ◇ Input shape of the CNN part
- ◇ Reshape input for LSTM

The default generated *dataset* considers input and label pairs of single input frames. The LSTM layer needs a sequence of frames for the training process, accordingly, the dataset was split into ordered sequences of frames. The ordered sequences were put into a training dataset and shuffled, this leads to the training usage of random frame sequences with different driving trajectories. The number of used frames for a single sequence can be set in a config file before the training process. Based on the observations of different training instances, the most promising sequence number was 4 as we already discussed in Section 3.2.2. The VPS system supports 30 Frames Per Second (FPS), depending on the necessary synchronization of the corresponding label for each frame, the recording FPS rate for the mounted camera was set accordingly. A sequence of 4 frames represents a timeframe of $\frac{4}{30}$ seconds which are ∼130 milliseconds.

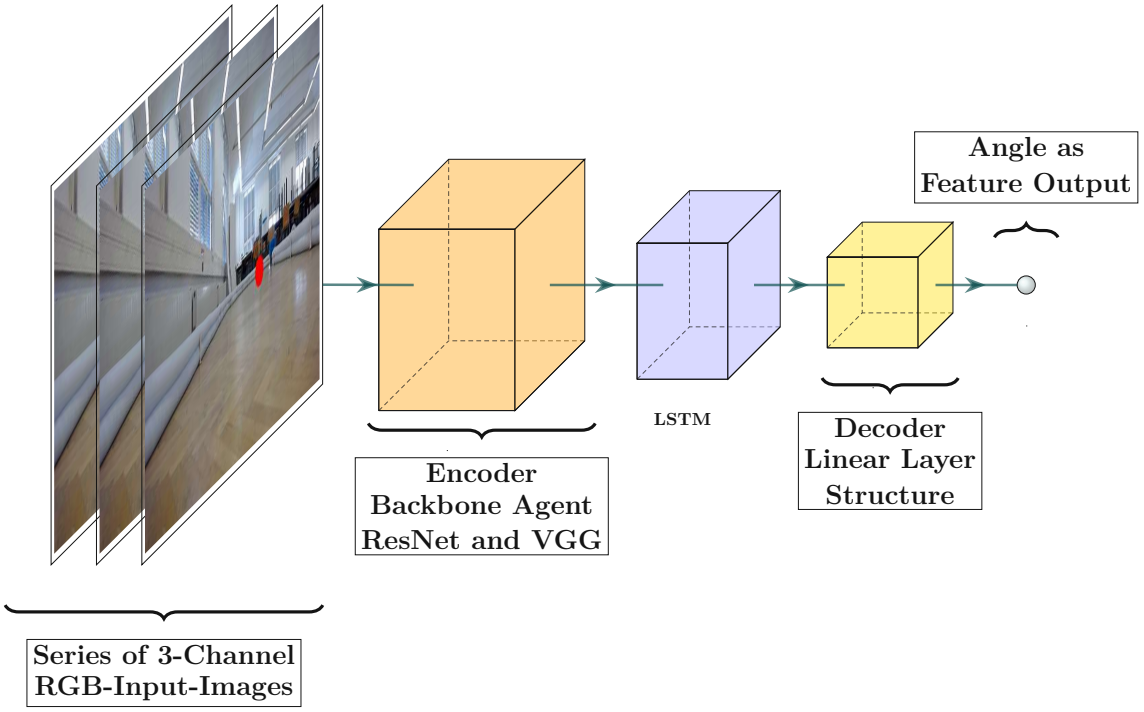


Figure 5.4: Agent model architecture with LSTM layer.

The tensor *input shape for the CNN part* has to support sequences of [B, S, C, H, W], where B is the batch size and S is the sequence length. C, H, and W are the channels, height, and width of the tensor. The CNN architecture for the pre-trained Pytorch models [34] expects an input of shape [B, C, H, W]. Therefore the batch is merged with the frame-sequence number from the input because the input shape has to fit the model's

input, which leads to [B * S, C, H, W]. With that change, the CNN architecture is considering the frame sequence simply as a bigger batch size and can do the feature extraction on all the input frames.

After the CNN structure the output of the pre-trained model has to fit the *input shape of the LSTM layer*. The classifier of the pre-trained models is used to flatten the input with linear layers to fit the shape and the number of input parameters for the LSTM. The output size of the LSTM layer considers the *hidden dimension* of this layer and has to fit the input shape of the following decoding part. This is similar to the single frame-based approach as the decoding part was kept in the same structure, it is a structure of stacked linear layers considering regularization techniques for a produced single value as output that serves as the steering angle. Depending on the fact if a current training process uses sequences or not the shapes of the model parts for both the encoding- and the decoding-structure had to be set correctly.

5.3.3 Evaluation Metrics

The evaluation of the agent model's training and testing part is based on the prediction of a float value for the steering angle. In comparison to a classification-based task for supervised learning, the evaluation has to consider this as a *regression* task. For a classification task, the prediction outcome represents a number of a fixed set of possible object outcomes, the prediction outcome will be directly compared to the label and if the prediction is correct there was 1 hit found. Unfortunately, for the prediction of float values, an exact comparison is not an appropriate evaluation method. For a regression task the aim is to learn a hypothesis based on the feature data points that are used during supervised learning [23]. The convention for the regression is to learn a hypothesis map for all data points that describes the smallest disparity between the prediction map and the points. The MSE loss is the function used to calculate the training loss all along the agent's training process.

As the regression task should use a fitting hypothesis with the smallest discrepancy an accuracy check has to consider that it is not about a direct comparison like a classification task. Therefore the implementation concerning the accuracy of the steering angle prediction takes into account a specific range of values that would be a valid prediction for the predicted value. Each of the predicted *angle* values is accounted as a correct prediction if it fits the specific label value range. For the angle prediction, the value can deviate 0.03 degrees from the angle label and will be still correct. The prediction is only counted as correct when the predicted value fulfills this specific value range.

5.3.4 Regularization

The training process of the agent model leads to overfitting for the default implementation without regularization techniques. Starting from the point of overfitting, regularization

techniques are needed to overcome this problem (as discussed in Section 2.4), therefore the dataset was extended with data augmentation and recordings from multiple tracks. Furthermore *dropout*-layers and *weight decay* for the optimizer of the learning process were used to avoid that the model overfits and provide a reasonable agent model with a converging training and validation accuracy.

Generally, the agent model should be focused on the track to predict reasonable driving commands according to the track course. The webcam that recorded the dataset saves the frames with a wider area. It does not only contains the track but also the surroundings and people, especially for the upper image of the frame. An idea to improve that fact is cropping the upper third so that the model will not be distracted during the learning process. Both of the images are shown in Figure 5.5, the image on the left shows the original input frame (see 5.5a), and the right image is the cropped version of that input frame (see 5.5b) to keep the agents focus on a smaller area of the original frame.

(a) Original input frame. (b) Cropped image.

Figure 5.5: Crop the upper third of the image to focus the track.

CHAPTER 6

Evaluation

This chapter discusses the results of both the attention-NN for the binary-mask-prediction of the raw input frame and the agent-NN for the steering-angle-prediction. The *human-attention* recorded via the VPS system and used for the dataset generation (see Chapter 3) can train an attention-NN (based on the UNet-architecture [41]) that makes reasonable predictions of a human's attention for the input frames of an F1/10-track. Therefore an important part of the work of this thesis was the created dataset that needed an adequate quality, otherwise, the attention-NN would not be able to learn a reasonable tendency where a human's attention is located for a specific frame.

Based on these attention-predictions the accuracy for the angle-predictions of the agent-NN (details of the agent are discussed in Chapter 5) were improved. Different NN implementations were done to compare the accuracy for different pre-trained models in the encoding part of the model as well as considering a LSTM layer with a frame-series input to the model. The discussion regarding these different implementations and the corresponding outcomes are shown in Section 6.2.1.

A highlight was the created workshop paper *Enhancing Robot Learning through Learned Human-Attention Feature Maps* [54] that got accepted for the ICRA Conference in May 2023 in London. The details on observations comparing different training budgets for the training and the impact on the difference for the Agent and Att-Agent on the performance is discussed in Section 6.2.2.

6.1 Attention Model Analysis

This section considers the results regarding the attention model prediction and compares the different model structures used and the corresponding advantages and drawbacks.

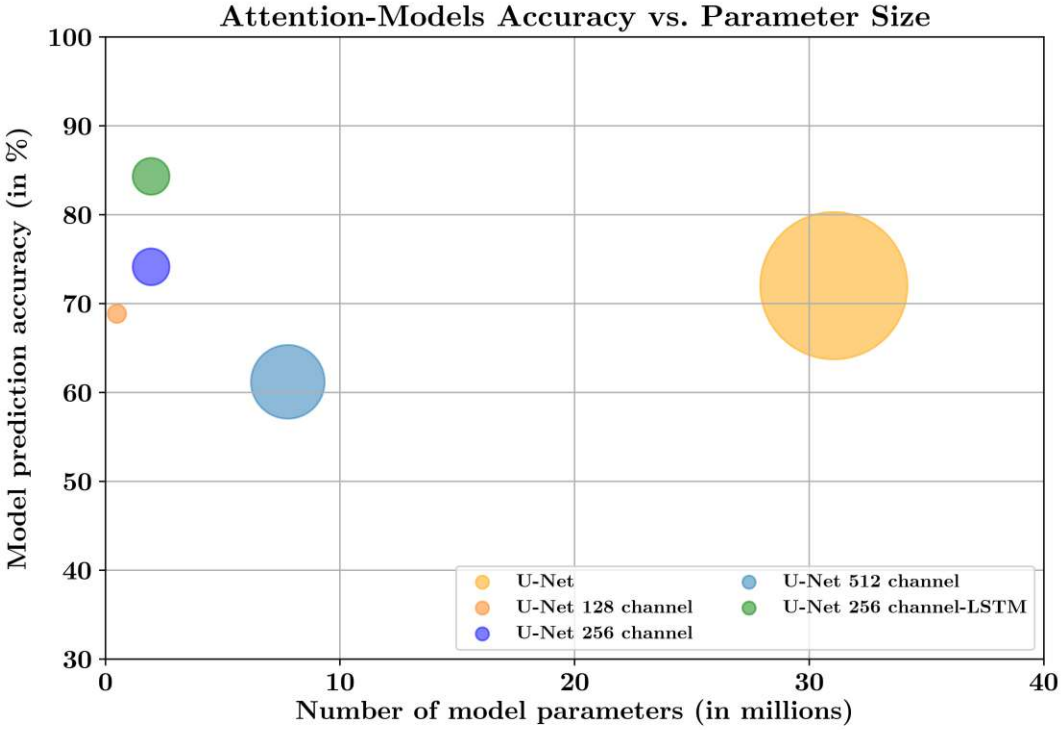


Figure 6.1: Attention Model comparison regarding accuracy concerning parameter size.

Starting from the default *U-Net architecture* [41] with 1024 channels (shown on the righthand-side of Figure 6.1) this implementation has the most features with more than 30 million model parameters. The number of model parameters correlates with the memory needed on the Nvidia Jetson Xavier NX and therefore this default implementation was not reasonable on the real F1tenth-Car (detailed explanation in the following Section 6.3). For performance improvement, the U-Net architecture was implemented with a channel size of 128, 256, and 512 to observe the outcomes with a smaller number of parameters.

Furthermore, an additional idea was to extend the U-Net architecture with a LSTM-layer to improve the model's consideration of previously received frames. This should smoothen the human's attention prediction and make the prediction more capable to avoid outliers in a sequence of frames to keep the human's attention more focused on specific driving areas on the track.

With smaller channel sizes it was able to reduce the overall training time of the models and increase the inference speed on the real F1tenth-Car for the attention model that is running on a ROS-node. The final version that was used on the car was the U-Net architecture with an integrated LSTM-layer (green colored in Figure 6.1). The accuracy with ∼85% was significantly improved compared to the other U-Net implementations and the number of parameters is still low, as the base model was the implementation with 256 channels (yellow below the green model in Figure 6.1). There are slightly more parameters for the U-Net with 256 channels that include the LSTM-layer compared to the model that uses single frames as input. But the jump in the accuracy of ∼10% justifies this small increase in the parameter number.

Still, it has to be considered that a NN that includes a LSTM-layer brings additional overhead to the computations on the Nvidia Jetson Xavier NX as the frame-sequence as input has to be prepared and a higher amount of data has to be transferred through the ROS-node and the model itself. This leads to a trade-off regarding the usage on the car as the most accurate attention-NN would be favorable for the best predictions for a series of frames, but when it comes to a lack of resources and the Nvidia Jetson Xavier NX starts to overwhelm the car may not be able to predict corresponding angles in a reasonable time and will not be able to manage the track.

Figure 6.2 shows the different performances of all U-Net implementations over the time of all 100 training epochs. The corresponding losses on the right visualize a drop within the first 40 epochs, then the training curves even out. The only model that is significantly able to further improve beyond the 40 epochs, is the U-Net architecture with 256 channels and the integrated LSTM-layer. Both the accuracy and the corresponding loss for this model are dominant to the other agent models. The main difference is the usage of a sequence of frames (visible in 3.10), necessary as input for the integrated LSTM. This leads to the fact that the model is trained more focused on a human-attention for following input frames, therefore it is more aligned to the trajectory that is driven by the car. Overall this improves the human-attention prediction by smoothening it and avoiding *outlier* areas as the predicted area is less jumping for following frames like it was trained on.

As discussed in Section 1.2.2, the observations on *human-attention* have shown, that the attention areas can occur really *jumpy*. These big adjustments for the attention coordinates position can even appear from one frame to the next. This property in combination with the fact of different attention areas that are shown for the same driving state for the real human driver, makes a correctness check quite challenging. Nevertheless, the task is interpreted considering a segmentation task, therefore a reasonable metric for the evaluation of the attention-NN prediction is discussed in Section 4.2.4. These outcomes illustrate, that the correctness is more about the tendency of predictions of the attention-NN for input frames and receiving reasonable sequences of following human-attention areas for sequential incoming frames.

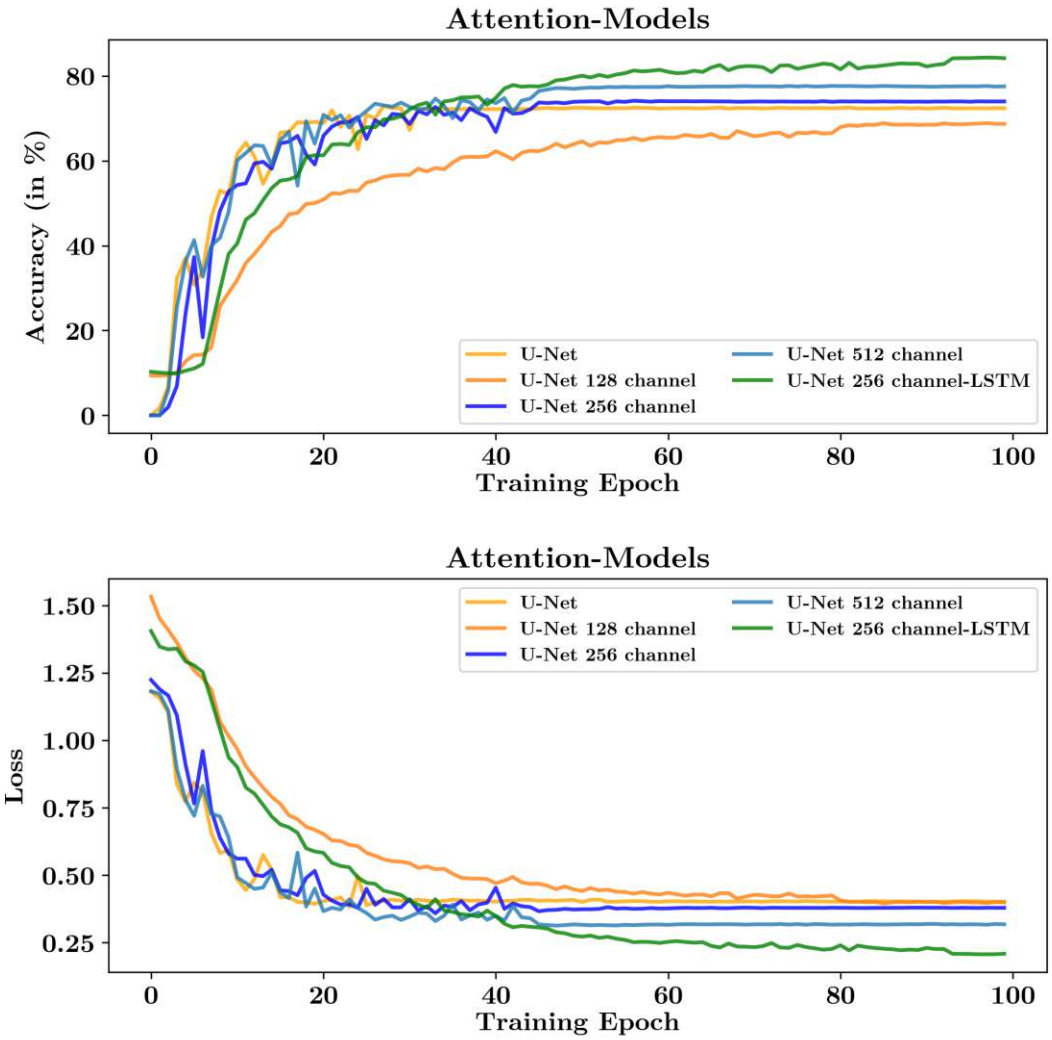


Figure 6.2: Accuracy/Loss for the epoch-training of the attention models.

6.2 Agent Model Analysis

In this section, the different variations of the implemented agent models are analyzed and compared to each other. The details regarding the architecture of the different agent implementations are discussed in Chapter 5.

Different pre-trained Pytorch model-versions [34] are used for the performance analysis of the model's predictions:

- VGG11 and VGG16
- ResNet18 and ResNet50

All of these models have a similar, but different model architecture considering the details. Generally, all these pre-trained models are quite common and reasonable for extracting the necessary features based on 2D input shape. Therefore these models are usable for this thesis approach, as the generated dataset (see Chapter 3) provides inputs that are recorded 2D input frames from the mounted camera of the F1/10 car.

The observations include the impact of the attention area on the performance of the neural network. Taking a specific neural network architecture with the same dataset and (hyper)parameters as the baseline, the model training has to be done with and without the marked attention area on the frames. The idea is to focus on the performance differences between the agent models regarding the usage of input frames with/without this human-attention area. Therefore, to distinguish between the implementations, a trained agent-NN that uses the human-attention feature got *Att-* as a prefix to make a clear difference.

As a reference for a default implementation, *Dave-2* [53] was used as the base for further comparisons concerning the improved accuracy. From this starting point the layer structure was made more sophisticated, the *Nvidia* Dave-2 *encoding-part* is implemented with a series of convolutional layers. This was replaced with different versions of pre-trained Pytorch models to make the structure more complex to extract more features from the input frames and create the ability to learn more with the same dataset. The *decoding-part* of the Dave-2 is a simple series of linear layers, the feature number of each layer was updated and the layers were extended by a Rectified Linear Unit (ReLU)-activation function and additional dropout-layers. With these measures, the prediction accuracy was improved significantly as the comparison of the different implementation variants is shown in the next Section 6.2.1.

During training, different implementation details were changed to get a best-working layer architecture that learns the most out of the provided features. To find the best hyperparameters a tool named *optuna* [18] (see Section 4.2.5) was used, therefore different value ranges for the parameters were specified and the tool trains the model for a specific number of epochs until it changes the parameters to find the best-fitting hyperparameters. With this, especially for the dropout layers the best fitting probability for the feature dropout was hard to find, as this is a sensible hyperparameter and there is the trade-off of overfitting if the dropout is chosen too high and the model does not have enough features to learn from.

For each of the models during the model-training different plots were created to analyze the behavior of the training curves regarding jumps and overfitting. This leads to the same design as already shown for the attention model in Section 6.1. A representation for the agent-NN with validation values for the accuracy and the loss is addressed in Figure 6.3 for the *Att-Resnet50* including a LSTM layer.

Figure 6.3 shows the overall best performing agent-NN, because the LSTM integrated to the *Att-ResNet50* brought a significant accuracy improvement. More important regarding the questions of this thesis is a comparison of the agent that uses the frames with human-

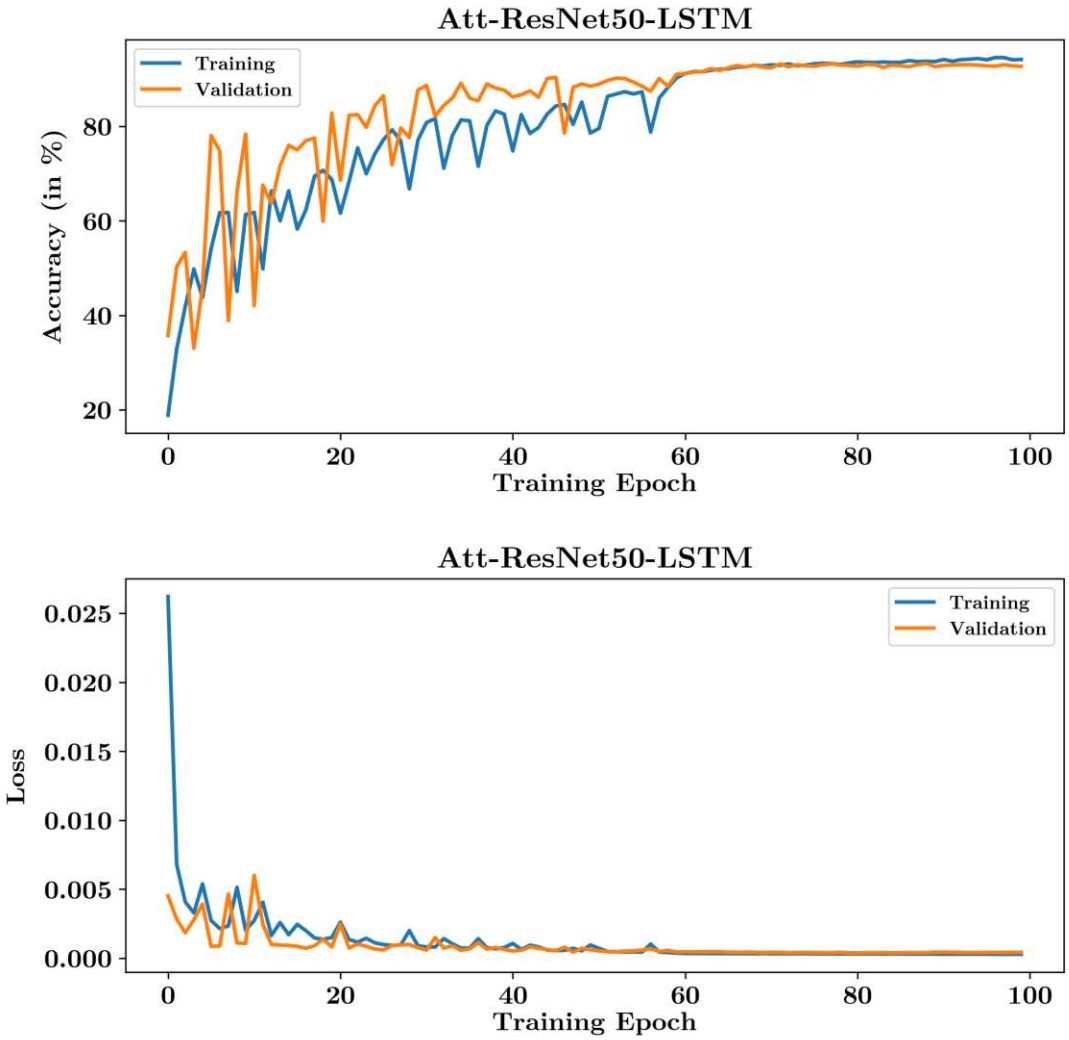


Figure 6.3: Accuracy/Loss for *Att-ResNet50* with LSTM.

attention during training compared to the agent that does not use human attention. Therefore for all the single framed implementations to the Att-Agent's another training run had to be done without the human-attention. The comparison for the *Att-Resnet50* and *ResNet50* is visible in Figure 6.4. A comparison of training curves with all other agent-NN implementations is shown in the next Section 6.2.1.

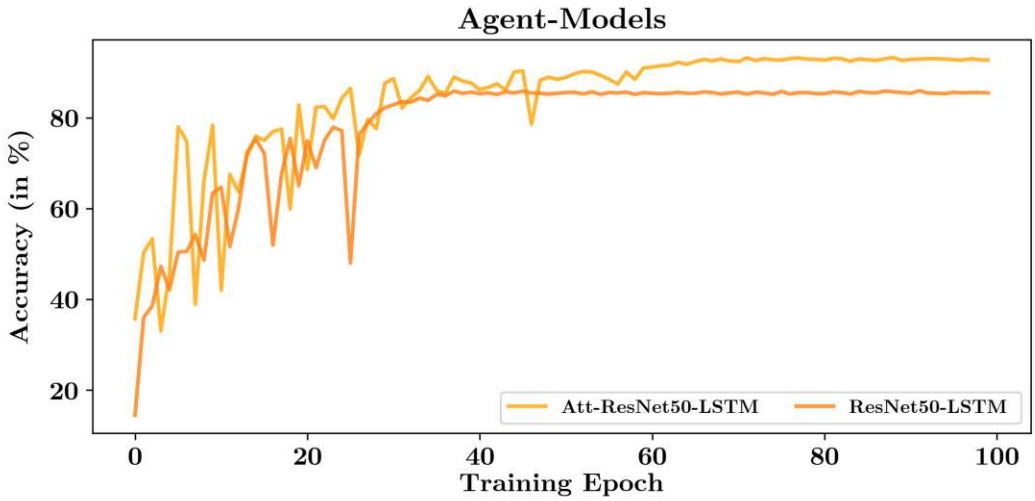
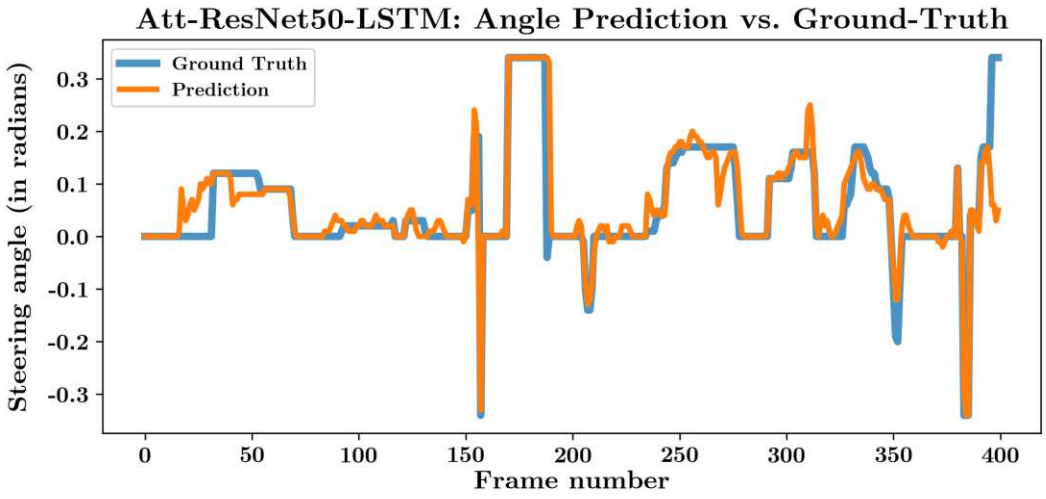


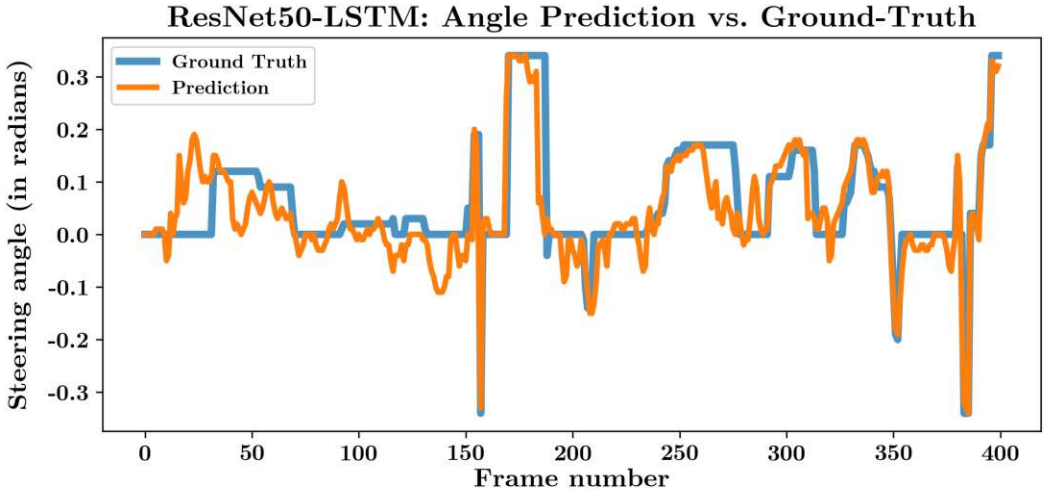
Figure 6.4: Accuracy for *Att-Resnet50-LSTM* vs. *ResNet50-LSTM*.

Additionally, as the agent-NN's were trained to predict the steering angle for driving on the track, the predictions of these angles were compared to the angle labels of the dataset part that was used for validation.

The following Figure 6.5 should put more focus on the difference between the trained agent-NN in respect to the usage of human-attention. For this plot a sequence of predictions was used to keep a more reasonable trajectory that represents a usual F1tenth-Track. The plot on top shows the *Att-ResNet50-LSTM* and overall the model is accurate with a few minor mispredictions. In comparison to that the bottom plot shows the agent implementation with *ResNet50-LSTM* that does not use human attention. On this plot for smaller angles, the predictions are fine, but an interesting observation is the fact that for curves on the trajectory where a bigger steering angle would be needed, the agent model does not perform so well at all. Therefore the model that uses the human-attention learns the tendency of the attention in curves and enhanced its performance compared to the agent-NN that does not use the human-attention. The human-attention area influences the agent-NN, from training it recognizes the attention area and this additional information is supportive for predicting a heading direction in the form of a steering angle. Additionally, the agent with the human-attention feature shows an overall lower volatility for the angle predictions addressed in Figure 6.5.



(a) Predictions vs labels for *Att-ResNet50* with LSTM.



(b) Predictions vs labels for backboned *ResNet50* with LSTM.

Figure 6.5: Predicted vs. ground truth steering angles.

6.2.1 Model comparison

The first part of the comparison should be based on the number of model parameters compared to the prediction accuracy that each of the different models achieves. Figure 6.6 visualizes the size of each model regarding the corresponding number of parameters and sorts them on the y-axis regarding the prediction accuracy. When the agent model was trained with the dataset that considers the human-attention feature, the *Att-* was added as a prefix to the models-name.

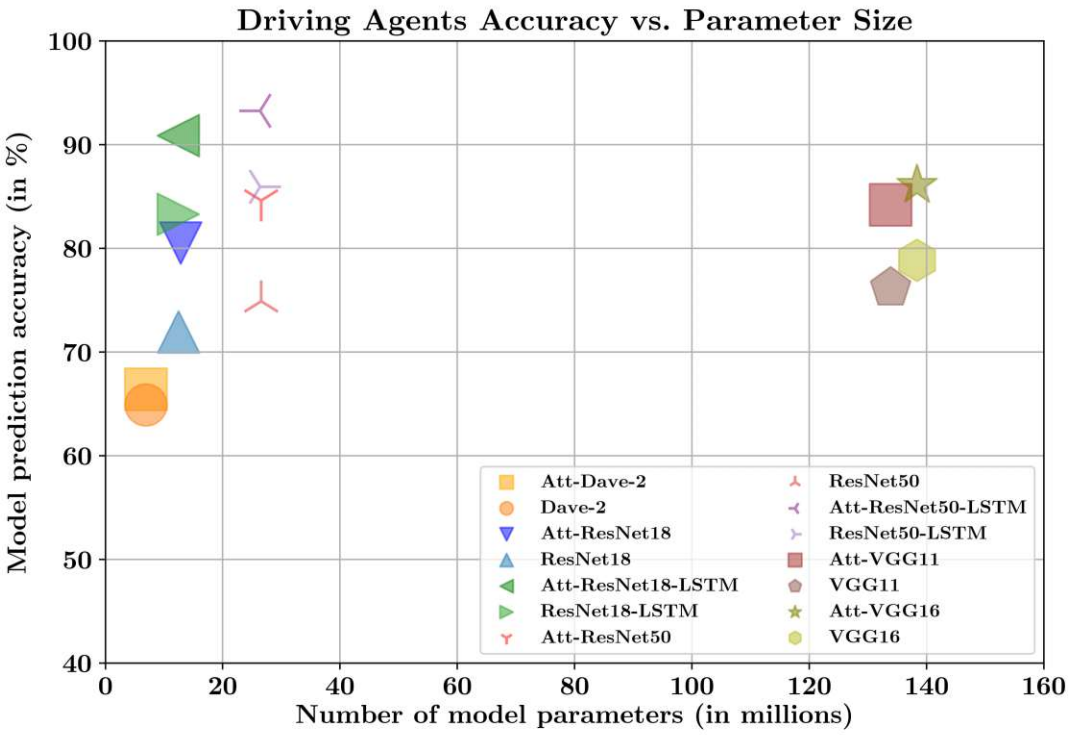


Figure 6.6: Agent-NN size vs. accuracy comparison.

The default implementations of the *Dave-2* and *Att-Dave-2* reach the lowest accuracy in Figure 6.6, the exact values are shown in table 6.1. From this starting point the agent-NN's that used *ResNet* as the pre-trained model in the encoding part has a significantly lower number of model parameters than the implementations that used *VGG* as the pre-trained model in the encoding part. The *ResNet* supported models are shown on the top-left side (see Figure 6.6). Even though the *VGG* supported models are shown on the top-right side of the figure with ∼6 times the size of the *ResNet* models, there is no real performance increase measurable with respect to accuracy.

With that observation concerning the limited resources on the Nvidia Jetson Xavier NX, the agent-NN that will be deployed for the steering angle prediction will use a version of the pre-trained *ResNet*. The lower amount of needed model parameters improves the overall inference performance on the real F1tenth-Car (details are discussed in Section 6.3). The *Att-ResNet18* has still ∼10 million parameters less than the versions with the *ResNet50*, therefore this version was used for the final result on the real car.

Comparing the agent implementations that use *VGG* as a pre-trained model, the different outcomes for *VGG11* and *VGG16* are shown in Figure 6.7b. All *VGG* models show a continuously slow increase in accuracy, there are still higher improvements till epoch 55,

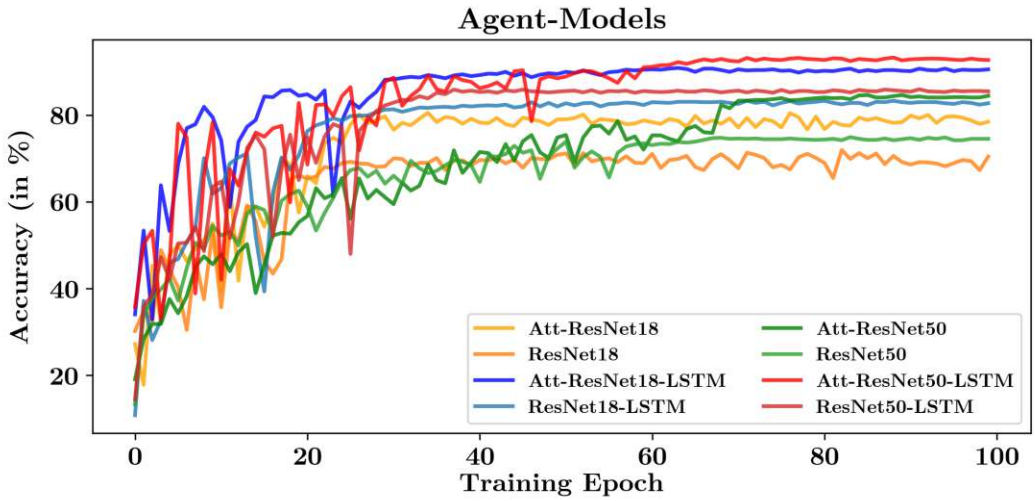
Models	*RNN*	*Accuracy in %* *Agent*	*Att-Agent*
Dave-2	✗	64.90	66.43
ResNet18	✗	71.96	80.53
ResNet18	✓	83.31	90.87
ResNet50	✗	74.92	84.62
ResNet50	✓	85.96	93.26
VGG11	✗	76.20	84.22
VGG16	✗	78.85	86.10

Table 6.1: Agent model accuracies.

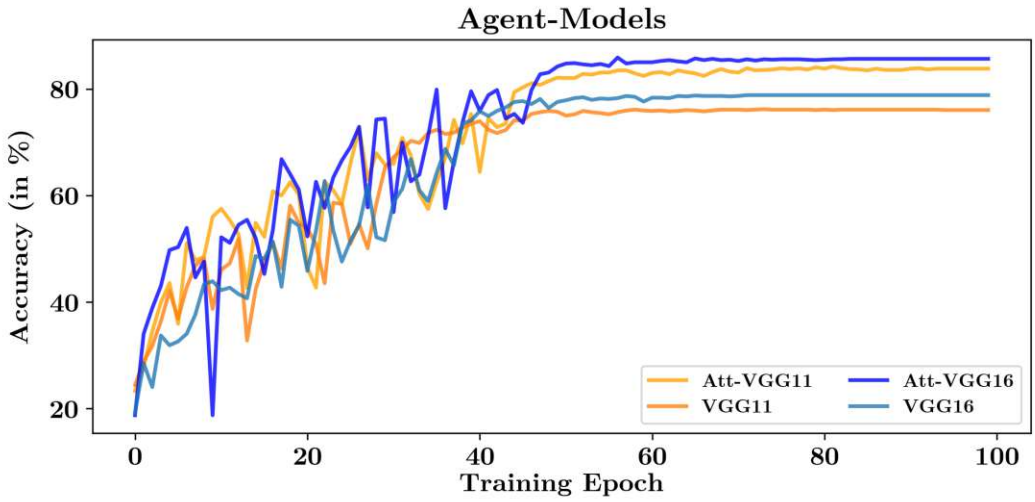
after that epoch it flattens and there is no further bigger increase. A clear outperformance regarding the accuracy is visible comparing the backboned *VGG11* with the *Att-VGG11* that uses the human-attention. The detailed accuracies are stated in table 6.1, where *Att-VGG11* dominates with ∼84% the model without attention. As the *Att-VGG16* has another ∼10 million of parameters more than the *Att-VGG11*, there is another bump in the accuracy, but with an even bigger drawback regarding the usage on the car.

Comparing these curves to the training curves in Figure 6.7 - showing the agent implementations using *ResNet* (Figure 6.7a) as a pre-trained model instead of *VGG* - most of the possible learning from the features is already done around epoch 40. This is a reasonable result, as the models that integrated *VGG* have a significantly higher number of model parameters as shown in Figure 6.6 and the model will need more epochs to reasonably learn the features for the input frames by making use of all that model parameters. The differences regarding the usage of the human-attention are similar to the findings of the *VGG* models above. Both variants with *ResNet* as a backbone for the agent implementations are outperforming the model that does not use human attention.

The top performance for the accuracy/loss combination is reached by the *Att-ResNet50* with LSTM, which is even able to surpass the 90% border. Overall a comparison for the backboned *ResNet* models including a LSTM layer is addressed in Figure 6.8. Considering the prediction accuracy for the angle, the models with LSTM are generally preferable compared to the models without LSTM. Table 6.1 contains the exact accuracies, e.g. the backboned *ResNet18* with LSTM without attention dominates the *ResNet18* with human-attention but without the LSTM. This circumstance states the advantageousness of the LSTM usage because the agent-NN does a better learning for the steering angle for the provided trajectory contained in the frame sequence. Unfortunately, for the hardware deployment addressed in Section 6.3.1, the usage of both the attention- and the agent-NN with LSTM would lead to a big delay between the prediction processing for the incoming



(a) Accuracy for *ResNet* as agent backbone.



(b) Accuracy for *VGG* as agent backbone.

Figure 6.7: Accuracy Comparison for *ResNet* and *VGG* models.

frames. Therefore only the attention-NN at the beginning of the prediction workflow (see Figure 6.11) was deployed with a LSTM layer for the ability to provide a reasonable FPS rate for the running F1/10 system on the car.

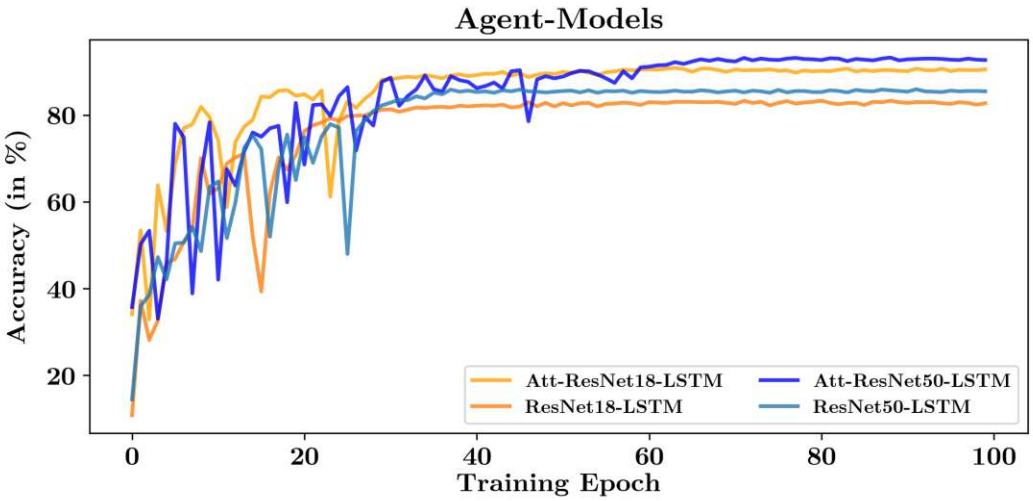


Figure 6.8: Accuracy Comparison for *ResNet* models including LSTM.

6.2.2 Dataset-Size Impact on Model-Training

The experiment done for a conference paper was focused on the agent-NN training concerning the usage of different sizes of the dataset. The dataset used for this experiment consists of ∼20,000 labeled frames and starting from the left x-axis entry 20% (see Figure 6.9) of all frames were used for the training. The size was increased to the full size of the dataset which is represented by 100% as the last entry on the x-axis.

The agent that was used for the different trainings was *Att-ResNet18*, whereas Figure 6.9 considers the model as Att-Behavioral Cloning (BC) and BC stands for the training without considering the human attention. The y-axis represents the MSE-loss for the model and in addition to the different dataset sizes on the x-axis that were used for the training, the plot shows a descending trend of the loss value with the increasing dataset-size.

The overall result shows that the size of the used dataset has a big impact on the MSE-loss of both agents. With the increasing amount of data, it is easier for the models to learn a complex relationship based on the fact that it has a lot more features to learn from. This leads to the loss-reduction for both models and it is visible in Figure 6.9, that the models trained with the highest amount of data (100% on the right side of the x-axis) showing the overall smallest MSE-loss.

Another observation can be made regarding the loss-difference between the *Att-BC* and the *BC*. Beginning with a smaller dataset size, there is a bigger outperformance for the *Att-BC* compared to the model that does not use the human-attention. With the increasing dataset size, at least for the full dataset size the difference in the MSE-loss is smaller for both models. This shows an interesting tendency: The usage of the

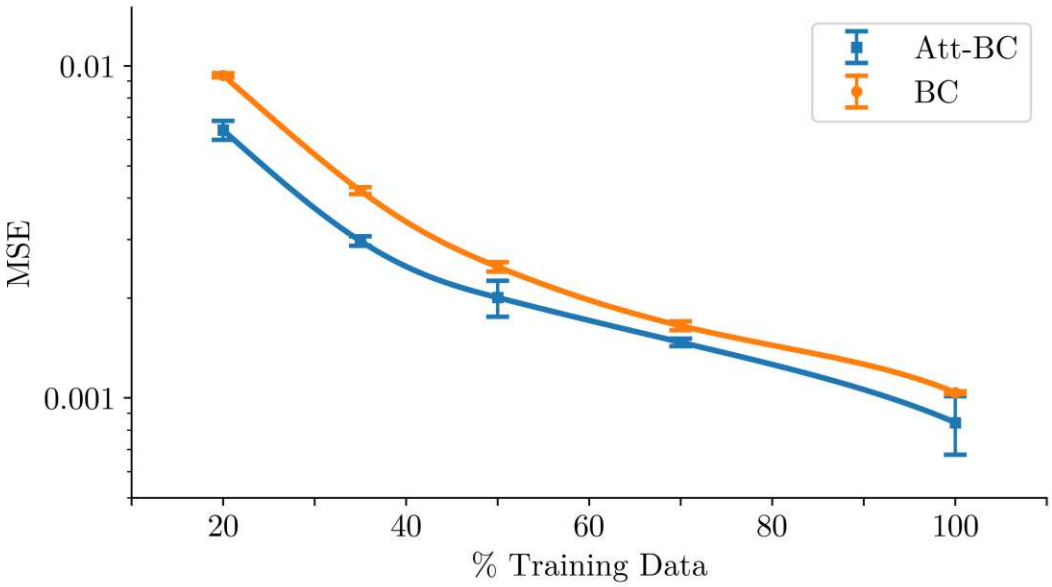


Figure 6.9: Loss comparison for different training budgets.

human-attention feature has a bigger impact on a model's prediction accuracy if a smaller amount of frames is available. With the increase in the dataset size, the loss difference gets smaller, as it can learn from a bigger amount of frames that capture a higher number of possible input scenarios. Therefore not only does the MSE-loss for the models reduces, but also the difference between both models themselves gets smaller and the bonus of the usage of human-attention gets smaller.

6.3 Simulation vs. Real Hardware Car

This section deals with the differences and challenges between the neural networks in simulation and on a real F1Tenth car. The following analysis is based on the prior result sections of the attention model (see Section 6.1) and the agent model (see Section 6.2). Often differences occur when comparing a simulation to running instances on real hardware. Therefore the aim of the thesis includes the deployment of both models the model trained with and without human-attention to check for a possible *sim-to-real gap*.

The idea was to build an F1/10-track and drive on it with everything set up on the real F1tenth-car to fulfill the overall aim of a steering angle prediction for the F1tenth-framework (see Figure 1.1). The predicted float value is directed to the Vedder Electronic Speed Controller (VESC) from the F1tenth-Framework, which is the controlling unit that is responsible for powering the car corresponding to the driving commands it receives. With that, everything was up and running on the real car and the car was able to

autonomously drive on the track for the predicted steering angle it receives as output from the agent-NN as the last part of the whole process flow on the car.

In the simulation it was possible to evaluate the agent prediction in a visualization as shown in Figure 6.10 where the *label angle (in yellow)* from the generated dataset is visually compared to the *predicted steering angle (in red)* value from the agent-NN output. Furthermore, the predicted human-attention area that was predicted from the attention-NN is illustrated as a red pixel area and the values of the agent-NN are dependent on that area.

Figure 6.10: Predicted (red) vs. label angles in simulation.

The performance observations on the real F1Tenth-car can be differentiated regarding the *accuracy* and the *inference speed*. Considering the speed performance the Pytorch framework was used as well for loading the trained attention- and agent-model. The webcam on the car and the VPS eye-tracking glasses that were used to generate the dataset run both on 30 FPS. Therefore the best performance may be achieved with the same FPS-rate. Both models would not perform reasonably on the CPU resources, therefore the GPU resources of the Nvidia Jetson Xavier NX [20] had to be used. For this necessary improvement, the running ROS-nodes got extended by the TensorRT-Software Development Kit (SDK) from NVIDIA [55]. With the model deployment, the following applications were running on the car during evaluation:

- F1tenth-Framework [15] for interpretation and usage of the driving commands on the car
- Deployment of both models (attention- and agent-NN) on ROS-nodes
- Recording of a ROS-Bag for evaluation purposes and the DAGGER-approach (see Section 6.3.2
- Logitech-Camera stream during driving

With TensorRT-API for Python [19] both models were running at a higher performance, but still, the usage of all the resources of the Nvidia Jetson Xavier NX was very high and it tend to overwhelm with all the running application. Therefore on the real car the achieved FPS-rate was *∼14 Hertz (Hz)*.

Besides the lower FPS-rate on the real car compared to the simulation, there was another observation considering the accuracy of the agent's steering angle predictions. As the

agent-NN was trained via supervised learning based on a dataset with *expert* driving commands, it performs well on areas that are recognizable to the environment and features of the training data. Tests on another track show, that there is a *drift* from the expert's trajectory. This can be explained by slight mispredictions for new input frames of non-trained areas. Continuously the error increases, until it comes to areas near the border that are nearly untrained and the car often crashes (see Figure 6.13).

To overcome this issue, the DAGGER algorithm stated in 6.1 was implemented on the real car. The details of this implementation are discussed in the following Section 6.3.2

The implementations and the training of the neural networks were done separately, but for the deployment onto the real F1Tenth car both of the models have to be connected for the possibility to predict driving commands on a raw input frame. The frames that are used for inference for both of the models will be put onto the Graphics Processing Unit (GPU) to increase the inference speed and for generating the possibility to reach enough predictions for each of the incoming frames. The steps needed to run both of the models on the real car are described in Section 6.3.1.

Therefore the next step is the pipeline creation from the input frame to an agent model prediction that will be used as driving commands on the real car. Figure 1.1 shows the process flow of a single frame input until the prediction outcome. Therefore several steps have to be performed sequentially (starting from left to right):

- Input frame is a single frame captured from the mounted camera on the F1tenth car and redirected to the attention model
- The attention model predicts an attention area based on the input frame
- Afterwards the frame with the predicted attention area is redirected to the agent model
- The agent model predicts driving commands based on the input frame with the attention area

For the implementation, both the attention and agent models were integrated into ROS-nodes for running them in simulation and on the real car. The prediction outcome of the driving commands is published on a specific topic that considers the steering angle for driving.

6.3.1 F1tenth Car Deployment

Besides the model evaluation in the simulation that was done for the attention and the driving command inference, observations are necessary on the real F1Tenth car for the best-performing models from the simulation. Therefore both models had to be integrated into ROS nodes to run within the F1/10-framework and provide driving commands for the driving systems. Accordingly, Figure 6.11 considers the whole work of this thesis from the beginning of incoming frames on the left that have to be put through the attention-NN to enrich the frame with the attention area in the first place. Afterward,

the enhanced frames have to be redirected to the ROS-node with a running instance of the agent-NN to predict the steering angle to operate the car on a track.

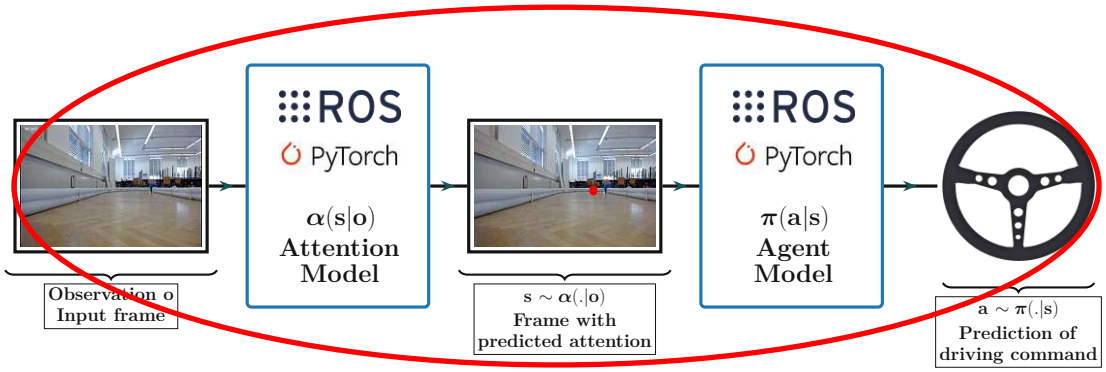


Figure 6.11: Attention-Agent prediction workflow for car deployment.

The whole ROS workspace is available on GitHub (thesis repository [52]) in the folder *racecar_ws*. The F1/10 system is used as the base to operate the car on the track. With that, it's possible to publish the steering angles to the specific ROS topic that is used from the F1/10-framework to interpret the values as driving commands. The *prediction_node* implements an ROS image callback that is triggered every time an incoming frame is published to the linked ROS topic of the callback. Additionally, the ROS node is extended by both neural networks, and the incoming frames are redirected for the sequentially happening 2D frame-based predictions. With that node, the necessary functionality for the whole process of the attention-agent prediction pipeline (see Figure 1.1) was successfully implemented on the car.

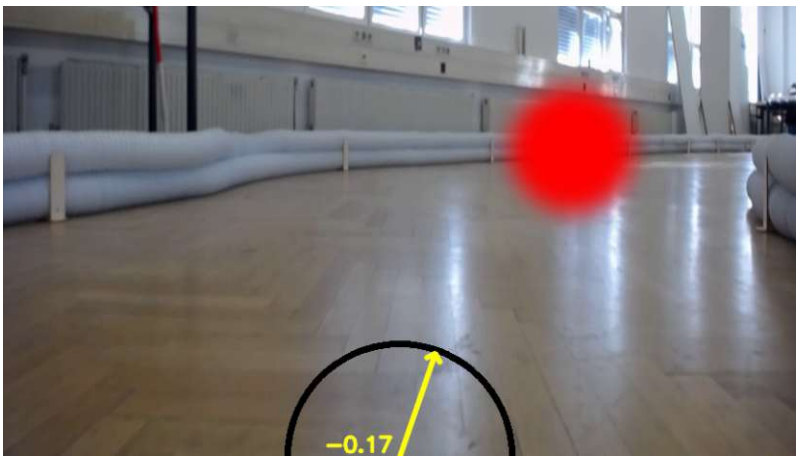
The first approach for the implementation of the *prediction_node* was done with the Pytorch framework, the models were loaded and put into the evaluation state. Unfortunately, with that implementation, the inference speed was too slow and this leads to the fact that the predictions were not provided within the necessary time frame. An improvement was achieved by the implementation of the TensorRT-SDK from NVIDIA [55], which is an engine approach based on a model in *Onnx* format. The established *Onnx* model was used for the engine generation via the Python-API [19]. With the integration of this engine for inference and the frame transfer onto the GPU, a performance boost for the inference speed was achievable and the predictions are done within a reasonable period for the input frames.

When starting the ROS nodes the *prediction_node* load the TensorRT engines or build them if necessary. This node contains the prediction for the attention area and the model predictions for the driving commands. Both predictions are published via ROS topics, whereas the driving commands are used for the F1Tenth system to steer the real car accordingly. The topic for the attention area is necessary to provide a visualization for observing the output on the car.

A sample frame of this visualization is shown in Figure 6.12. The attention area is the

red-marked pixel of the specific frame and the value shown on the bottom is the predicted steering angle with the corresponding arrow that points in the steering direction.

The inference performance of the attention-agent prediction pipeline is important, both models are starting their prediction (*prediction_node*) based on a callback that redirects the input frame to the models. The incoming frames have a rate of 30 FPS and therefore the best case is a prediction rate that would be capable of this FPS rate. Especially for checking the combination of inference speed and accuracy the implemented visualization is handsome. It is immediately visible if the prediction is accurate and reasonable for a car's state and that the prediction does not take too long.



(a) Negative steering value.

(b) Positive steering value.

Figure 6.12: Visualization of human-attention area and steering angle for an incoming frame.

6.3.2 DAGGER (Dataset Aggregation)

The ML-training for the agent-NN is done by supervised learning (for details see Section 2.2). The idea of this approach is that the agent should learn by mimicking the expert's behavior.

The issue of the data distribution mismatch for supervised learning had to be resolved to compare the models reasonably to each other on a real track. The problem arises from the fact, that the model has to do predictions for areas that have never been seen similarly. As shown in Figure 6.13, the accumulated error drifts too far from the expert trajectory and as a result, the model cannot correct the aggregated error and collides into the border. From the generated dataset based on the recorded data from real human drivers, this area especially directly nearby the border was not considered. The human drivers were driving pretty much in the middle of the track most of the time. Hence, the dataset lack of angle labels as well as human-attention area label coordinates for this area near the border. Because in the first place, it additionally has a slight impact on the attention-NN that cannot do the inference for the human-attention prediction with the same confidence as for other areas. Therefore a frame with a lower-quality human-attention area is forwarded to the agent-NN and it gets even harder for it to make a reasonable steering angle prediction.

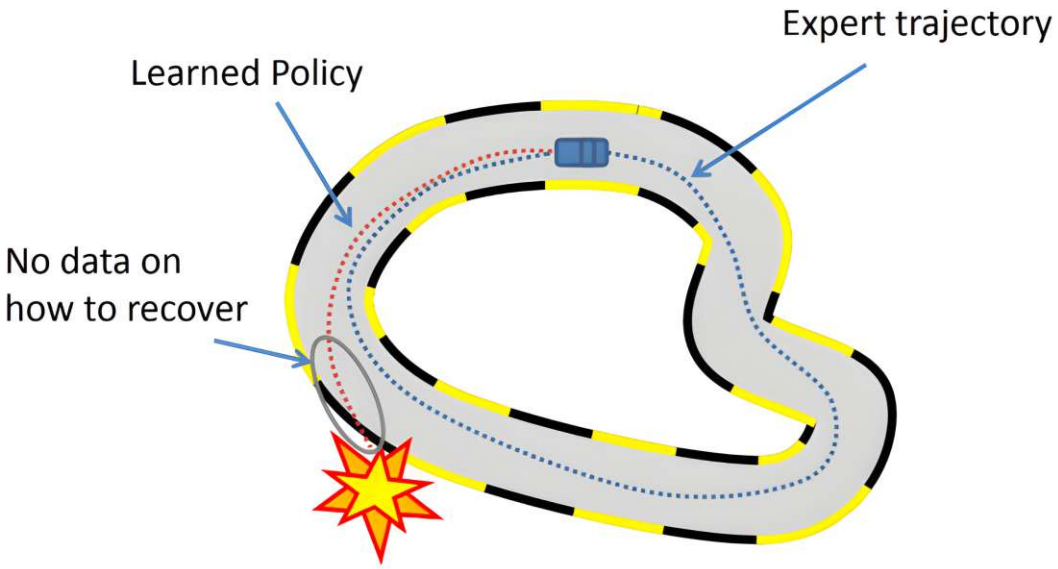


Figure 6.13: Distribution mismatch for the supervised learning approach from Liu.

The idea to resolve this issue is to additionally train the model for these areas near the border as well. The approach that is capable to do this supervised-model improvement is called DAGGER [57]. The general idea is to retrain an existing model multiple times but

increase the dataset by newly created frames for areas that still create problems until the model improves enough and can easily handle the situations it was planned for.

In algorithm 6.1 the DAGGER approach is described regarding the description from Ross et al.. The starting point is the original dataset that should be extended with data for the critical sections on the track, a number for the iterations that the DAGGER-approach should take, and the existing policy π of the trained agent-NN. But this policy π is extended with an expert, because when it comes to areas where the prediction of the trained model is too far from an expert decision, *the expert's decision has to be considered instead of the prediction to get reasonable labels for the newly created dataset.* Then for each of the iterations, the DAGGER algorithm has to consider a few steps. First, with the existing policy π additional data has to be recorded during the model's runtime. Based on the prediction accuracy, either the predicted value or if a correction is needed, the value of the expert's action has to be stored as a label. With these values an additional dataset D_i has to be created for the current iteration of the algorithm. This newly created dataset has to be concatenated with the original dataset. Furthermore with this bigger dataset a new policy π_i has to be trained for the current iteration. In the last step, the existing policy π will be updated with the newly trained policy of the iteration.

Algorithm 6.1: DAGGER Algorithm.

1: **Inputs:** Original dataset D_0, number of iterations N, existing policy π
2: Policy π is a combination of the trained agent-NN and the expert policy π^*
3: **for** $i = 1$ to N **do**
4: Collect new data with policy π in the environment
5: Generate dataset $D_i = \{(s, \pi^*(s))\}$ with states from π and necessary actions from the expert policy π^*
6: Concat the datasets: $D = D \cup D_i$
7: Train a new policy π_i on the aggregated dataset D
8: Update the policy: $\pi \leftarrow \pi_i$
9: **end for**

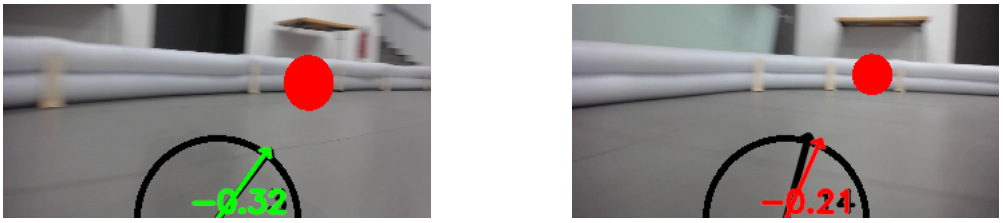
The algorithm in 6.1 represents the theoretical process for the general DAGGER approach that can be used for all neural networks and the necessary datasets. For the use-case of the F1tenth-car the DAGGER algorithm was implemented with the trained agent-NN and the corresponding original dataset as starting point. That means the trained model is the policy π as stated in algorithm 6.1, for the expert decision π^* a Follow The Gap (FTG) was implemented. FTG is a simple approach that uses the scan information from the car-mounted LiDAR to consider all obstacle distances to find the biggest gap and calculate a correct angle to reach this gap.

A single retraining iteration on the track considering the DAGGER-approach consists of the following steps:

- ◇ Record a ROS-Bag from driving the car on the track – considering predicted vs. expert angle
- ◇ Use the ROS-Bag to create additional frames with angle labels according to predicted/expert angles
- ◇ Concat the 'new' dataset to the original dataset
- ◇ Retrain the model on the new bigger dataset

The angle calculation with FTG is considered as an expert decision, if the agent neural network predicts an angle that is too far away from the expert's decision, a correction is needed, and the FTG angle is recorded instead of the predicted angle. A visualization for this is done in Figure 6.14, where the left Figure 6.14a shows a valid prediction (marked green) that is nearly exactly overlying the expert angle. Additionally, it is visible that the human-attention is considered as the arrow of the angle prediction directs to the red pixel attention area. On the right side in Figure 6.14b, the predicted angle is more than 5 degrees off the expert angle, and therefore the arrow of the predicted angle is marked red.

With the DAGGER implementation it was possible to overcome the issues that arise from the trained agent-NN that only uses supervised learning.



(a) Valid angle prediction within the range.

(b) Invalid angle prediction with deviation from the expert.

Figure 6.14: Visualization of expert vs. predicted angle on the real car.

6.3.3 Domain-Shift

The agent-NN had to be tested on another track and the angle visualization of Figure 6.14 was recorded on the real car in front of the *Informatikhörsaal*. With the trained agent-NN based on supervised learning this leads to the distribution mismatch as already discussed in Section 6.3. Hence, the DAGGER-algorithm was used directly on the track and the approach was executed for 3 iterations. The results for these 3 iterations are shown in Figure 6.15. This was a lot of effort, starting from the left side with the default entry on the x-axis, this is the correction comparison for the expert's actions needed for the already trained models *Att-Agent* and *Agent*. During the *default* run, a ROS-Bag was recorded for storing the predicted- or expert-angle to each of the incoming frames.

Based on this ROS-Bag, an additional dataset was created to increase the amount of data for unseen areas. The generated dataset with input-frame and angle-label pairs was concatenated to the original dataset. Then the 1st retraining for both models the *Att-Agent* and the *Agent* was done. In detail, the additional data had to be uploaded to a TU Vienna server to have enough GPU computation power for the retraining. After several hours the retraining was done and the following iterations were done in the same way as explained for the 1st iteration.

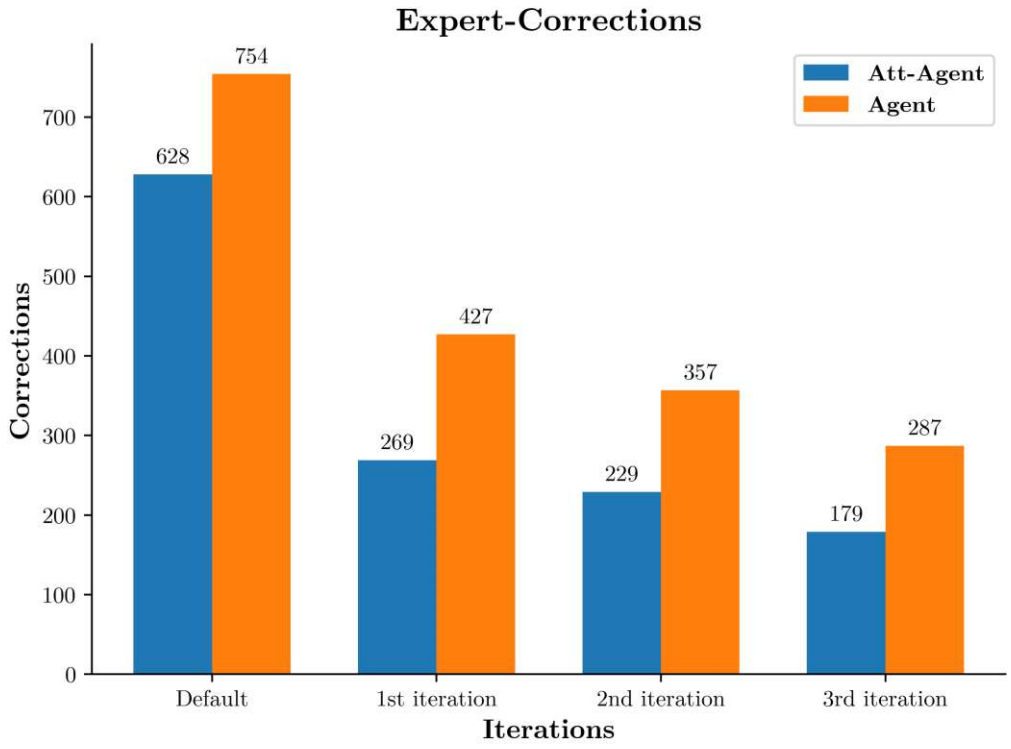
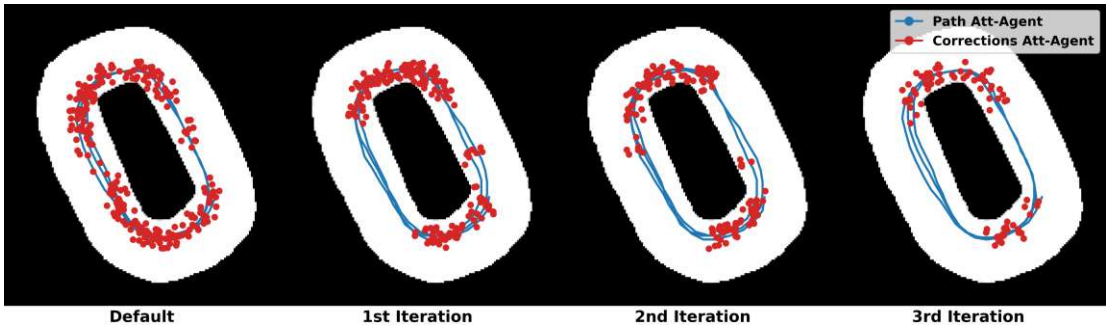


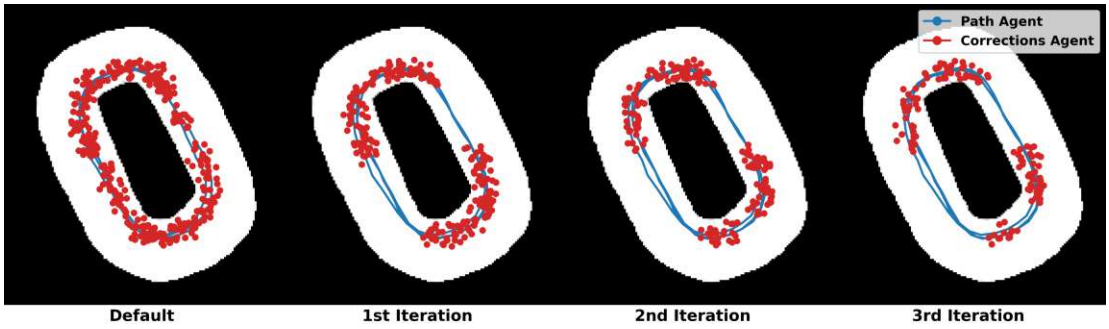
Figure 6.15: Dagger iterations show an improvement for the needed corrections.

Even though Figure 6.15 shows that the number of corrections is continuously reducing for both models, the overall number of corrections is less for the *Att-Agent* that considers attention. As shown in Figure 6.15, with additional iterations (increasingly using frames of *unknown* areas), the agent-NN improves the accuracy of the steering angle. On the x-axis, the increasing number of iterations shows a correlation to the reduction of the necessary corrections on the track. The DAGGER-approach works for both models the *Att-Agent* that uses attention and the *Agent* model that does not use attention. A further observation is the even lower number of corrections for the *Att-Agent* compared to the *Agent*. This highlights again the improvement by the usage of the *human-attention* during training, as for each of the iterations the *Att-Agent* is still better performing on the track than the *Agent*.

Additionally, as Figure 6.15 only considers the number of corrections needed, Figure 6.16 uses a path visualization of the track that was used for the experiment. This gives interesting information about where on the track most of the corrections happened. Therefore the information on the number of corrections is enhanced by the location on the map where all of these corrections happen for both models. The upper image shows the reduction of the corrections for the *Att-Agent*. In the same way, this is done for the bottom image for the *Agent*. It is visible that the *Att-Agent* has less corrections compared to the *Agent*. This analysis shows, that for this track the *Agent* has still a high number of corrections for the 2nd and 3rd iterations, especially on the short straight before entering into the curve.



(a) Corrections needed for the *Att-Agent*.



(b) Corrections needed for the *Agent* without attention usage.

Figure 6.16: Comparison of needed corrections for the agent model with/without attention usage.

Nevertheless, this shows that the *Att-Agent* considering attention outperforms the accuracy on the real track with a real F1tenth-Car. This model needs less corrections from an expert's action for all 3 iterations. Generally, this experiment confirms that the DAGGER-approach can improve the agent-NN behavior on the track. The number of corrections is continuously dropping and the agent-NN is therefore able to make good steering angle predictions to drive on the track.

CHAPTER 7

Conclusion and Outlook

In this thesis, we have studied the impact of *human attention* in the learning of end-to-end policies, and demonstrated how its integration in the training pipeline shows a positive impact.

The necessary parts of the workflow depicted in Figure 7.1 are:

- Data collection (marked in *blue*)
- Design of Attention predictor (marked in *red*)
- Design of Agent model (marked in *green*)

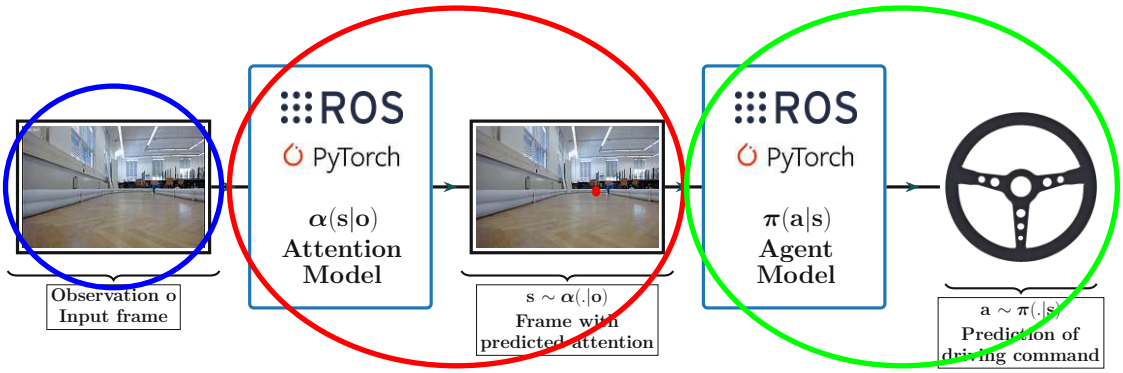


Figure 7.1: Attention-Agent prediction workflow.

By integrating human attention into the learning process, we enhance the training of the agent NN by introducing an additional human-attention feature in the form of a binary mask which highlights the focus area on the input frames. The result for the agent NN shows that, considering human-attention, the learning is facilitated by the correlation between the *steering command* and focus areas. In the following, we review the RQs and summarize the main findings.

RQ0 *How can we obtain a rich enough labeled dataset for human attention in driving situations?*

To answer this question, we develop a data-aquisition system on top of a *F1tenth car* using the *VPS* to record the human gaze. With this setup, we collect a large amount of recordings of *human-attention* and driving commands from the human driver. The details on the data-generation process are reported in Chapter 3. As in supervised learning, the quality of the collected data highly affects the performance of the NN model training. If the labeling of the input samples is accurate, the model predictions will present a sufficient quality to be used to autonomously control the vehicle. However, if this is not the case, the performance of both models will degrade as the quality of data. To address this challenge, we devise considerable effort in the first part of the project, to collect a comprehensive and diverse dataset.

RQ1 *How much is a neural network's training-and-prediction performance improved by the attention mechanism?*

Based on the generated dataset, we design a NN model that is capable to predict the area where the human is most-likely to focus. The implementation is based on the *U-Net* [41] architecture. Details and evaluation of the attention-model are discussed in Section 6.1.

Finally, we train a second NN model, the agent, to predict the steering command to drive the car on the track. We compare the agent trained with and without attention features, and explore various backbone models. Section 6.2 reports the complete analysis and comparison. In conclusion, we confirm a positive impact on performance for the model enriched with human-attention features, and quantitatively evaluate the improvement to answer RQ1.

RQ2 *What differences are observable and measurable between the simulation and real car performance?*

Having validated the proposed approach in simulation, we consider deployment in the real world to answer RQ2. We port the model on the on-board Nvidia Jetson Xavier NX [20], optimize for GPU- and memory usage, and compare the performance. To mitigate the worsening of performance due to the sim-to-real transfer, as outlined in Section 6.3, we integrate the DAGGER-algorithm for online data-collection. In conclusion, we observe and quantify the performance loss to answer RQ2, in term of resource consumption, FPS-rate and prediction accuracy.

RQ3 *What is the robustness of the trained agent to domain shift (e.g. different track)?*

In the last experiment, we test the agent on a different track and assess the cross-track robustness to answer RQ3. We compare again the performance of the agent with and without attention, and assess the reduction of corrections utilizing the DAGGER approach, as described in Section 6.3.2. After this analysis, we conclude that both the

agents show sensitivity to sim-to-real and cross-track transfer, however, the introduction of online data collection proved to be an effective solution to mitigate these issues.

As a closing remark, our study on training with human attention under limited amount of data, has been reported in the paper *Enhancing Robot Learning through Learned Human-Attention Feature Maps* [54], accepted at the ICRA 2023 workshop RAP4Robots.

7.1 Future Work

In future work, there are few directions to further enhance our work. While the use of behavioral cloning for autonomous driving demonstrate to be accurate on in-distribution data, our study highlighted the sensitivity to out-of-distribution scenarios (e.g., unseen track) and how the model can be trained in a robust way with iterative online data collection. In this direction, a combination of supervised pretraining and online finetuning with reinforcement learning may be a promising direction to explore. Moreover, other design choice to model observation and actions (e.g., multi-step predictions of controls) may lead to better results than end-to-end training.

A second research direction would look at efficient deployment under limited hardware resources. Apart from the performance improvement due to updated software and hardware technologies, the dependency on two models for attention and control prediction respectively, represent the main computational bottleneck. A possible solution consists of merging the two models and design a unique architecture that predicts attention and control inputs using multiple heads. On one hand, jointly training a model to predict attention as auxiliary feature may still retain the advantage of using human attention, as training with auxiliary tasks improved deep learning applications. On the other hand, the use of a single model would make sensible use of resources, allowing better parallelization through GPU.

In conclusion, the assessment strongly suggest that incorporating human attention into learning pipelines can enhance the performance compared to models trained without this feature. By further exploring the use of human attention, we can develop safer, more efficient and reliable autonomous driving systems. This research offers promising directions towards human-inspired capabilities of autonomous navigation systems and safer intelligent transportation for all.

List of Figures

List of Tables

List of Algorithms

Glossary

Agent Agent neural network that was trained without the human-attention. 57

Att-Agent Agent neural network that includes the human-attention feature for the training. 57, 62

F1/10 F1Tenth is the general system that is used to operate the real car. 2, 4–7, 9, 10, 28, 32, 47–49, 57, 61, 67, 69, 71, 72, 83

Human Attention Frame Is an input frame where the human-attention area is a marked area with red pixels. 6, 40, 41, 49, 51

ICRA Conference IEEE International Conference on Robotics and Automation. 57

RAP4Robots ICRA workshop: Representations, Abstractions, and Priors for Robot Learning. 81

Acronyms

AI Artificial Intelligence. 2, 12
ANN Artificial Neural Network. 15
API Application Programming Interface. 9, 70, 72

BC Behavioral Cloning. 68

CNN Convolutional Neural Network. 4, 11, 15–17, 20, 49–54

DAGGER Dataset Aggregation. 9, 70, 71, 74–78, 80, 85
DARPA Defense Advanced Research Projects Agency. 2
DL Deep Learning. 2, 3, 6, 11, 12, 14, 15, 19–21

FC Fully Connected. 52
FPS Frames Per Second. 53, 67, 70, 73, 80
FTG Follow The Gap. 75, 76

GPU Graphics Processing Unit. 71, 72, 80, 81

Hz Hertz. 70

IMU Inertial Measurement Unit. 48
IoU Intersection over Union. 44, 45

KDE Kernel Distribution Estimation. 30, 31, 83

LiDAR Light Detection and Ranging. 3, 48, 75
LSTM Long Short-Term Memory. 4, 17–19, 33, 34, 39, 43, 48–50, 52–54, 57–59, 61–64, 66–68, 83, 84

ML Machine Learning. 1–3, 5, 12, 14, 15, 20, 22, 23, 37, 74, 83
MLP Multi-Layer Perceptron. 15, 20
MSE Mean Squared Error. 22, 54, 68, 69

NLP Natural Language Processing. 4
NN Neural Network. 2–9, 11, 13, 14, 17–23, 26, 28, 29, 33, 34, 37, 38, 40–45, 47, 49–52, 57, 59, 61–63, 65–68, 70–72, 74–80, 83, 84

ONNX Open Neural Network Exchange. 9

ReLU Rectified Linear Unit. 61
RL Reinforcement Learning. 20, 83
RNN Recurrent Neural Network. 3, 4, 11, 16–20, 49, 66
ROS Robot Operating System. 7, 9, 28, 29, 49, 51, 59, 70–72, 76, 77
RQ Research Question. 2, 3, 79, 80

SDK Software Development Kit. 70, 72

VESC Vedder Electronic Speed Controller. 69
VPS View Point System. 5, 7, 26–28, 38, 53, 57, 70, 80

Bibliography

[1] M. Kyriakidis, J. de Winter, N. Stanton, T. Bellet, B. Arem, K. Brookhuis, M. Martens, K. Bengler, J. Andersson, N. Merat, N. Reed, M. Flament, M. Hagenzieker, and R. Happee, "A human factors perspective on automated driving," *Theoretical Issues in Ergonomics Science*, 03 2019. 1

[2] G. Neto, "From single-agent to multi-agent reinforcement learning: Foundational concepts and methods learning theory course," 2005. 1

[3] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, apr 2020. [Online]. Available: http://arxiv.org/abs/1910.07738 2, 11, 15, 16, 49

[4] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. K. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *CoRR*, vol. abs/2002.00444, 2020. [Online]. Available: https://arxiv.org/abs/2002.00444 2

[5] C. Chen, T. Wu, Z. Guo, and J. Cheng, "Combination of deep neural network with attention mechanism enhances the explainability of protein contact prediction," 2020. 2

[6] A. Makrigiorgos, A. Shafti, A. Harston, J. Gerard, and A. A. Faisal, "Human visual attention prediction boosts learning & performance of autonomous driving agents," *CoRR*, vol. abs/1909.05003, 2019. [Online]. Available: http://arxiv.org/abs/1909.05003 2, 4

[7] A. Sharif and D. Marijan, "Evaluating the robustness of deep reinforcement learning for autonomous and adversarial policies in a multi-agent urban driving environment," 2021. 3

[8] A. Mohammadhasani, H. Mehrivash, A. F. Lynch, and Z. Shu, "Reinforcement learning based safe decision making for highway autonomous driving," *CoRR*, vol. abs/2105.06517, 2021. [Online]. Available: https://arxiv.org/abs/2105.06517 3

[9] M. R. Bachute and J. M. Subhedar, "Autonomous driving architectures: Insights of machine learning and deep learning algorithms," *Machine*

Learning with Applications, vol. 6, p. 100164, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827021000827 3

[10] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State-of-the-art and challenges," *CoRR*, vol. abs/2009.14349, 2020. [Online]. Available: https://arxiv.org/abs/2009.14349 3

[11] A. Sharif and D. Marijan, "Evaluating the robustness of deep reinforcement learning for autonomous and adversarial policies in a multi-agent urban driving environment," 2021. 3

[12] Y. Bengio, A. C. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR*, vol. abs/1206.5538, 2012. [Online]. Available: http://arxiv.org/abs/1206.5538 3

[13] X. Wang and Y. Yang, "Neural topic model with attention for supervised learning," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 08 2020, pp. 1147–1156. [Online]. Available: https://proceedings.mlr.press/v108/wang20c.html 3

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf 4

[15] F1Tenth. (2022) F1tenth - build documentation. [Online]. Available: https://f1tenth.org/build.html 4, 7, 9, 70

[16] OpenCV. (2022) Opencv - largest computer vision library. [Online]. Available: https://opencv.org/ 7

[17] Pytorch. (2022) Pytorch - machine learning framework. [Online]. Available: https://pytorch.org/ 7, 21

[18] Optuna. (2022) Optuna - a hyperparameter optimization framework. [Online]. Available: https://optuna.org/ 8, 45, 61

[19] NVIDIA. (2022) Nvidia tensorrt python api reference. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/api/python_api/ 9, 70, 72

[20] Nvidia. (2022) Jetson xavier nx. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/ 9, 70, 80

[21] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *Sn Computer Science*, vol. 2, 2021. 12, 15, 16, 17, 18, 19, 20, 83

[22] Z. Zhou, *Machine Learning*. Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-981-15-1967-3 13, 14, 83

[23] A. Jung, *Machine learning : the basics*, ser. Machine learning: foundations, methodologies, and applications. Gateway East, Singapore: Springer, 2022. 14, 15, 20, 22, 54, 83

[24] Z. Li, W. Yang, S. Peng, and F. Liu, "A survey of convolutional neural networks: Analysis, applications, and prospects," 2020. [Online]. Available: https://arxiv.org/abs/2004.02806 15, 16, 83

[25] L. Wiest. (2023) Recurrent neural networks - combination of rnn and cnn. [Online]. Available: https://collab.dvb.bayern/display/TUMlfdv/Recurrent+Neural+Networks+-+Combination+of+RNN+and+CNN 17, 19, 83

[26] A. T. (k21academy). (2023) Recurrent neural networks (rnn) tutorial: Rnn training, advantages & disadvantages (complete guidance). [Online]. Available: https://k21academy.com/datascience-blog/machine-learning/recurrent-neural-networks/ 17, 83

[27] C. C. Aggarwal, *Neural Networks and Deep Learning*. Cham: Springer, 2018. 17, 21, 22, 23, 45

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 18

[29] C. Olah. (2023) Understanding lstm networks. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ 18, 19, 83

[30] J. Duan, S. E. Li, Y. Guan, Q. Sun, and B. Cheng, "Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data," *IET Intelligent Transport Systems*, vol. 14, no. 5, pp. 297–305, feb 2020. [Online]. Available: https://doi.org/10.1049%2Fiet-its.2019.0317 20, 48

[31] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," 2016. [Online]. Available: https://arxiv.org/abs/1611.05397 20

[32] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html 20, 83

[33] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," 2019. [Online]. Available: https://arxiv.org/abs/1911.02685 21

[34] Pytorch. (2022) Torchvision.models. [Online]. Available: https://pytorch.org/vision/0.8/models.html 21, 50, 53, 60

[35] C. C. Aggarwal, *Artificial Intelligence - A Textbook*. Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-72357-6 21, 50

[36] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org. 23

[37] Pytorch. (2022) Transforming and augmenting images. [Online]. Available: https://pytorch.org/vision/main/transforms.html 26, 34

[38] S. Ulmer, D. Scheuchenstuhl, and F. Resch, "Gitlab repository of the dataset," https://github.com/CPS-TUWien/learning_human_attention, 2023. 26

[39] Pytorch. (2022) torchvision.transforms. [Online]. Available: https://pytorch.org/vision/0.9/transforms.html 35

[40] ——. (2022) torchvision.models. [Online]. Available: https://pytorch.org/vision/0.8/models.html 35

[41] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. [Online]. Available: https://arxiv.org/abs/1505.04597 38, 39, 43, 57, 58, 80, 83

[42] G. W. Lindsay, "Attention in psychology, neuroscience, and machine learning," *Frontiers in Computational Neuroscience*, vol. 14, 2020. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncom.2020.00029 38, 41, 44

[43] H. Zhang, Q. Lian, J. Zhao, Y. Wang, Y. Yang, and S. Feng, "Ratunet: residual u-net based on attention mechanism for image denoising," *PeerJ Computer Science*, vol. 8, p. e970, May 2022. [Online]. Available: https://doi.org/10.7717/peerj-cs.970 38

[44] X. Wang and Y. YANG, "Neural topic model with attention for supervised learning," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 1147–1156. [Online]. Available: https://proceedings.mlr.press/v108/wang20c.html 38

[45] Nvidia. (2022) Getting start with u-net industrial. [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/resources/unet 39

[46] S. Ulmer, "Tu vienna gitlab repository containing the machine learning part for attention," https://gitlab.tuwien.ac.at/cyber-physical-systems/lehre/thesis/attention/attention, 2022. 40

[47] Pytorch. (2022) Lstm. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html 43, 52

[48] J. Wang, X. Zhang, P. Lv, L. Zhou, and H. Wang, "Ear-u-net: Efficientnet and attention-based residual u-net for automatic liver segmentation in ct," 2021. [Online]. Available: https://arxiv.org/abs/2110.01014 44

[49] W. Wang, K. Yu, J. Hugonot, P. Fua, and M. Salzmann, "Recurrent u-net for resource-constrained segmentation," 2019. [Online]. Available: https://arxiv.org/abs/1906.04913 44, 45

[50] Z. Zhang. and J. Ohya., "Movement control with vehicle-to-vehicle communication by using end-to-end deep learning for autonomous driving," in *Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods - ICPRAM,*, INSTICC. SciTePress, 2021, pp. 377–385. 48, 52

[51] S. Jonah, "Prediction of steering angle for autonomous vehicles using pre-trained neural network," *European Journal of Engineering and Technology Research*, vol. 6, no. 5, pp. 171–176, Aug. 2021, this journal provides immediate open access to its content on the principle that making research freely available after publication on the EJ-ENG website to the public supports a greater global exchange of knowledge. 49

[52] S. Ulmer, "Master thesis repository," https://github.com/staypoked/da, 2022. 50, 72

[53] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 04 2016. 50, 51, 61

[54] Stefan, D. Ulmer, F. Scheuchenstuhl, L. Resch, R. Berducci, and Grosu, "Enhancing robot learning through learned human-attention feature maps," https://drive.google.com/file/d/1vSq1PLrVwEk-1Cpfr3J6dy2G1mv9B8df/view, 2023. 57, 81

[55] NVIDIA. (2022) Nvidia tensorrt documentation. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html 70, 72

[56] S. Liu, "Introduction to dagger," https://shuijing725.github.io/files/20190926_DAgger.pdf, 2019. 74, 84

[57] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 627–635. [Online]. Available: https://proceedings.mlr.press/v15/ross11a.html 74, 75