Mathematisch-Naturwissenschaftliche Fakultät

Lernbasierte Computer Vision

Masterthesis

Visual Abstractions for Autonomous Driving

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Lernbasierte Computer Vision
Micha Schilling, micha.schilling@student.uni-tuebingen.de, 2021

Bearbeitungszeitraum: 1.10.2020 - 30.4.2021

Betreuer/Gutachter: Prof. Dr. Andreas Geiger, Universität Tübingen
Zweitgutachter: Prof. Dr. Hendrik Lensch, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Micha Schilling (Matrikelnummer 4232764), April 30, 2021

Abstract

Semantic segmentation is commonly used as an intermediate representation in autonomous vehicles since it encodes all the relevant information about the environment. This leads to an improvement in the performance of the driving policy as well as other auxiliary benefits like reduction in variance due to different training seeds. However, including too many different classes in the segmentation increases the labeling costs and has been shown to sometimes even adversely affect the driving performance of the agent. Therefore, it is imperative to learn a good intermediate representation that only contains information relevant to the downstream driving task. Towards this goal, this work proposes a novel approach using a genetic algorithm to automatically extract the relevant classes for such an intermediate representation. The method is validated by analyzing the importance of the extracted classes for the task of urban driving in challenging multi-town multi-weather evaluation settings in the CARLA simulator. Furthermore, this work examines the extent to which low-dimensional affordance representations can be used in addition to semantic segmentation as a visual abstraction to increase the performance and interpretability of a trained model. Finally, an empirical study provides practical insights into important metrics and hyperparameters for training and evaluating driving policies. The insights from this analysis are used to train an autonomous agent which is able to outperform recent state-of-the-art imitation learning methods.

Acknowledgments

First of all, I would like to thank Prof. Dr. Geiger and Prof. Dr. Lensch for making this work possible and allowing me the freedom to try out my own ideas. Furthermore, I want to thank Kashyap Chitta and Aditya Prakash for their support, for the countless constructive discussions, and for answering all of my questions throughout the past months. Finally, I want to thank my family and my girlfriend for their continuous support, encouragement, and snacks.

Contents

1 Introduction

The potential of autonomous vehicles is huge and they are most likely going to revolutionize cityscapes and transportation as we know them today [1]. The number of accidents and traffic fatalities could be reduced drastically since the vast majority of accidents in streets are caused by human error [2]. At the same time, the comfort of traveling would improve. Additionally, self-driving cars would enable people who currently are not able to drive a car to be much more mobile.

Based on all the prospects of autonomous cars, it is not surprising that large companies are spending billions of dollars in development [3] to be among the first to produce driverless cars. In recent years, the rapid progress in deep learning and computer vision has led to major breakthroughs in the development of self-driving technology. However, at the time of writing, the development of fully autonomous vehicles still remains a hard and unsolved problem. Meanwhile, media interest in the topic is very high and news of accidents involving partially autonomous vehicles are spread all over the world[1].

A vehicle that is to drive autonomously needs sensors to perceive its environment and a processing unit that processes the signals from the sensors and uses them to generate control commands. While the choice of a suitable sensor suite for autonomous driving is a major discussion point[2], there is no doubt that cameras are among the most important and most frequently used sensors. Not only because they are inexpensive, but also because people use their eyesight to navigate in traffic as well and therefore the current infrastructure is designed precisely for this purpose.

To be able to drive autonomously, the processing unit needs a thorough understanding of the ego-vehicle's environment. Therefore the driving task is typically split into a perception and a control part. In the perception part, the relevant information is extracted from the sensor data and transformed into an intermediate representation which is easier for the control algorithm to understand. The control algorithm then processes this information to output suitable actions (steering angle, throttle, and brake). However, reliably abstracting the sensor data in such a way that only the relevant information is preserved in the intermediate representation is a difficult problem that has yet to be resolved.

[1] https://www.tagesschau.de/wirtschaft/technologie/usa-tesla-unfall-103.html
[2] https://towardsdatascience.com/why-tesla-wont-use-lidar-57c325ae2ed5

This work aims to analyze different segmentation-based intermediate representations and the effect of these visual abstractions on autonomous driving and thereby contributes to the further development of self-driving cars.

1.1 Contributions

The key goal of this work is to investigate the effect of different visual abstractions on the performance of end-to-end trained autonomous driving agents. The contributions of this thesis are:

- A novel approach to automatically determine relevant classes in semantic label maps used as intermediate representation for a given task.
- The validation of this approach by assessing the performance of an autonomous vehicle trained on semantic label maps and evaluated in the CARLA simulator.
- An empirical study to provide insights regarding the training of autonomous vehicles using imitation learning.

1.2 Thesis Outline

This thesis is structured as follows: Chapter 2 introduces important concepts and related work and gives a brief overview of its development up to the current state of research. Chapter 3 describes the implementation details of the autonomous driving agent, as well as a description of the algorithms and improvements developed. The detailed description of the experiments performed in the CARLA simulator, as well as an analysis of their results and conclusions, can be found in chapter 4. Finally, chapter 5 contains the conclusion with a summary of this work's findings.

2 Preliminaries

In this chapter, the preliminaries and most important related work are described. Section 2.1 provides a brief overview of the history of autonomous driving and establishes related concepts. In section 2.2 the CARLA simulator is introduced. The simulator provided the platform for the training and evaluation of all models in this work. The models are trained with imitation learning, which is described in section 2.3. The end of the chapter (section 2.4) is a brief introduction to semantic segmentation, the technique used for visual abstraction in this thesis.

2.1 Autonomous Driving

Not long after cars started being mass-produced, people began to dream of driverless vehicles. One of the earliest mentions of truly autonomous cars goes back to the Futurama exhibit at the 1939 World's Fair, where GM sketched cars driving by remote control on highways that contained electromagnetic circuits [4]. Later several prototypes were developed using this technique [4]. Although some prototypes were successful, this method has never prevailed in everyday cars as it is not practical to equip a lot of roads with such a technology. Therefore it is necessary to develop autonomous vehicles that are able to use the already existing infrastructure. In the 1980s, Ernst Dickmann and the Navlab team independently developed prototypes of self-driving vehicles that were capable of driving on public roads for the first time. Both teams refined their approaches in the following years [5]–[7]. Between 2004 and 2007, the DARPA Grand Challenge[1] led to further progress in the development of autonomous vehicles. In the subsequent years, many automotive manufacturers, tech corporations, and new startups started doing research and development in the field of autonomous driving and great progress has been made. However, contrary to earlier predictions by some of these companies, there are still no fully autonomous vehicles to date.

The hardest part of solving autonomous driving are the last few percentages to full autonomy. This is due to the long tail distribution of different scenarios: while it is relatively easy to solve the most common scenarios like following a clearly defined road on a sunny day, in day to day traffic there are many rare situations that need to be solved reliably, to make fully autonomous driving possible. Building robust models that perform well in these situations turns out to be a very hard problem.

[1] https://www.darpa.mil/news-events/2014-03-13

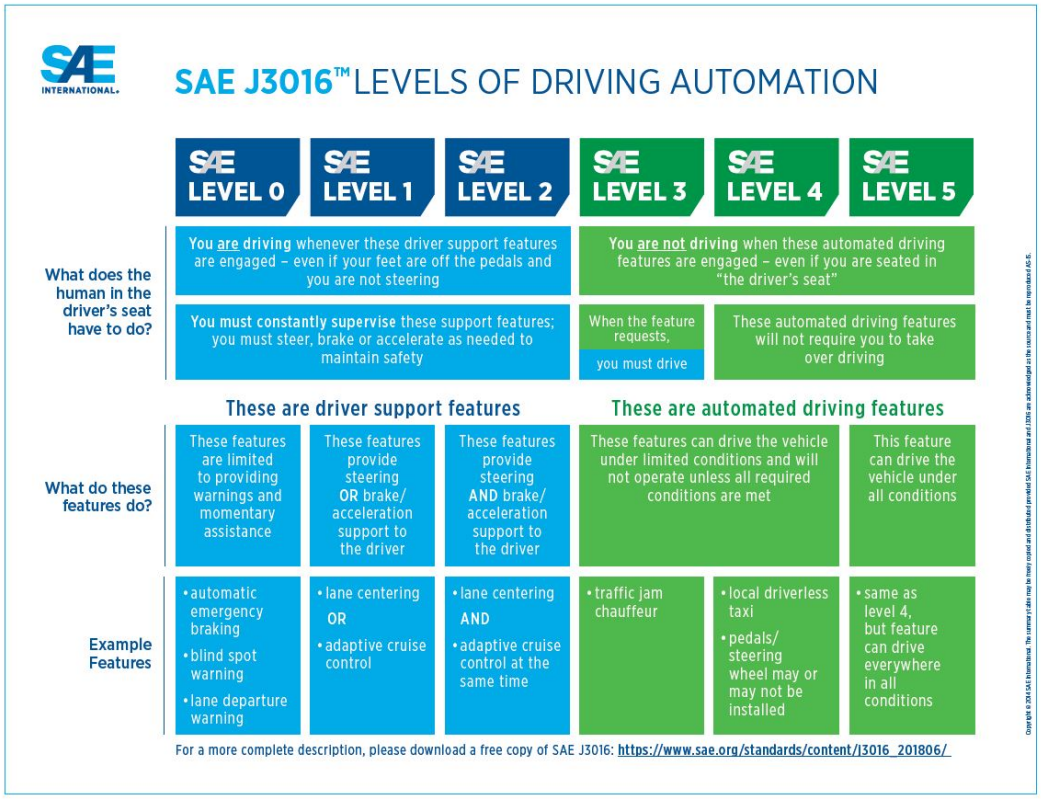


Figure 2.1: SAE levels of driving automation, graphic provided by SAE [8].

The Society of Automotive Engineers (SAE) has developed the six levels of driving automation, a widely adopted categorization for autonomous driving (see Figure 2.1). It ranges from level 0 (no automation) to level 5 (full automation). Modern cars often have built-in driver assistance systems like lane- and distance-keeping. These technologies correspond to level 3 automation. Those vehicles are able to autonomously navigate simple scenarios like traffic jams, highway driving, or automatic parking, but the driver must be able to intervene and take back control at any time. Today, Waymo, a subsidiary of Alphabet Inc., is among the leading companies and one of the only corporations to test level 4 automation in public road traffic. They currently operate a publicly available fleet of fully driverless cars in a very restricted area [9]. This restriction simplifies the problem significantly as the area is thoroughly examined and therefore there are far fewer unpredictable situations. Overall, level 4 and level 5 automation remain difficult to realize.

The different approaches used in the development of autonomous vehicles can be divided into two types: the modular pipeline and the end-to-end learning approach [4].

- **Modular Pipeline**:
  The modular pipeline consists of multiple modules, through which the information flows from the sensory input through 'low-level perception', 'scene parsing', 'path planning', and finally through the 'vehicle control'. The benefit of this approach is that it is way more interpretable compared to the end-to-end approach. If the agent fails, it is possible to analyze the information that was passed in between all the modules and to identify the point of failure. Additionally, it is possible to combine machine learning with traditionally designed algorithms designed with expert knowledge like traffic rules. This can be an important safety feature, because the output of a traditional algorithm may be easier to predict. Another benefit is that many teams could work independently from each other on separate modules as long as the communication between those modules is defined properly. The problem is that the human-defined intermediate representations, used between the modules, might not encode all relevant information in an efficient representation. Given the team sizes and safety standards, this approach is mainly used in the industry.

- **End-to-End Learning**:
  As the name suggests, in the end-to-end approach the driving policy is defined by a single function, typically a neural network. It is trained directly on the sensory input and outputs the controlling commands. Therefore the model combines the tasks of perception, planning, and control. The benefit of this approach is that it is much simpler to implement than the modular approach. Currently, the problem with this approach is that it is basically a black box and therefore it is hard to interpret why the network takes a specific action. Moreover, the fact that state-of-the-art deep neural networks are highly susceptible to adversarial attacks [10] shows that it is very hard to assess the robustness of such networks.

No matter the approach, machine learning enabled progress in autonomous driving and according to Drago Anguelov, head of research at Waymo, its influence in the field will increase further[2]. This work builds on the recent advancements in modular approaches and end-to-end learning and the waypoint model (see section 3.3) could be considered a hybrid model combining the benefits of both.

[2] https://youtu.be/Q0nGo2-y0xY

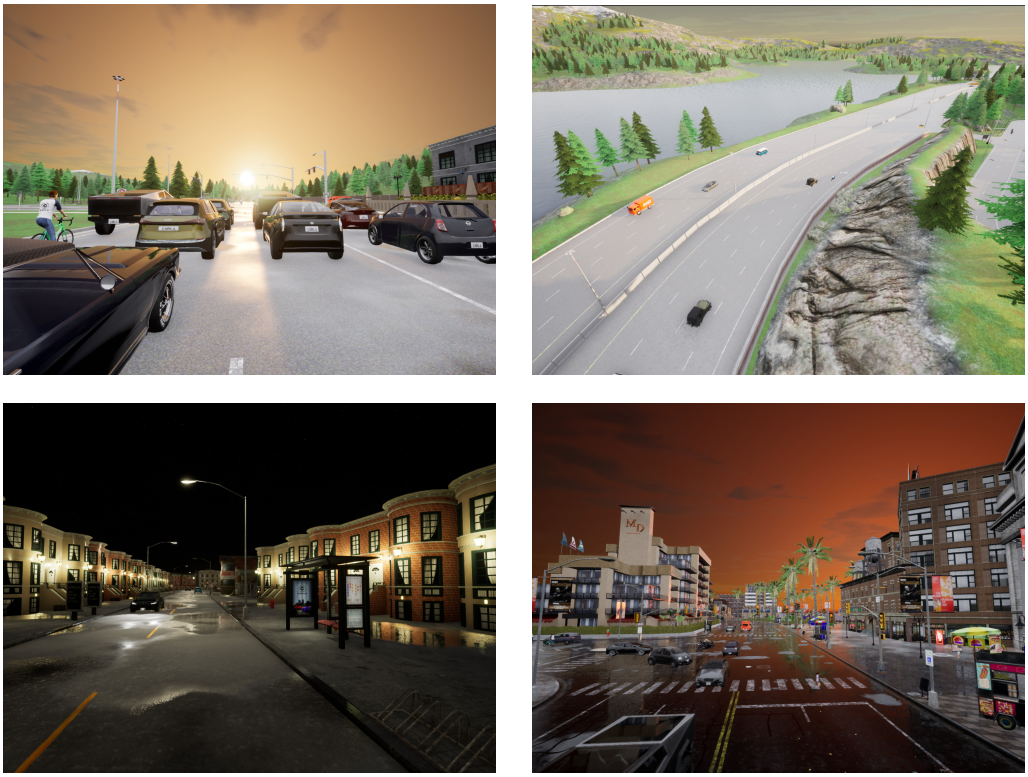


Figure 2.2: Screenshots of a few examples of different weather conditions and environments in the CARLA Simulator: blinding sunset in Town06 (top left), highway in Town04 (top right), residential area at night in Town02 (bottom left) and busy intersection at dusk in Town10 (bottom right).

2.2 CARLA Simulator

CARLA [11] is an open-source driving simulator, specifically designed for autonomous driving research. It is developed using Unreal Engine [12] and has an easy-to-use python API to control the simulated world as well as the cars driving in this virtual environment. It features eight different towns where the cars can drive and complex driving scenarios[3] can be placed along the route. Additionally, traffic density and weather can be customized by the user. This can be used to simulate difficult conditions for the agent to master. Weather customization is especially important for the perception part of self-driving vehicles since conditions like a wet street on a rainy day, dense fog, or a low standing and blinding sun can change the appearance of the scene drastically. The CARLA simulator features 10 continuously adjustable parameters to generate a wide variety of different weather conditions and times of day (see Figure 2.2).

[3] `https://leaderboard.carla.org/scenarios/`

Compared to the real world, driving and testing a self-driving agent in a simulator has several benefits:

- Running a simulation on a computer to test an autonomous vehicle is much cheaper than fully equipping a real car with all necessary sensors and driving around the real world. Furthermore, accidents in the simulator have no consequential costs, but they can indicate errors in the architecture of the self-driving car.
- Simulations allow generating difficult scenarios, like a pedestrian unexpectedly running across the street, and test them without danger of harming anyone. Such situations are very rare in real life, but for fully autonomous driving the vehicle needs to be able to act appropriately. Additionally, the simulator can help to understand the driving behavior when the same scenario is run multiple times with minor variations.
- The simulator can speed up the feedback cycle, as it is faster to evaluate an agent in the simulator than in the real world.
- Moreover, it is possible to acquire perfect ground truth data from the simulation. This not only includes the speed limits, street signs, and exact position of all traffic participants, but also information that is very laborious to obtain for a real-world dataset. Examples of this are ground truth semantic segmentation labels and perfect bounding boxes. Furthermore, any number of sensors can be easily mounted anywhere on the vehicle.

However, using simulators has disadvantages as well. Since simulators are simpler than the real world, there will always be a gap between the simulator and the real world and models trained in a simulator do not easily generalize to the latter. The physics in the simulator will always be a little bit off and there are always situations in the real world that the simulator does not handle correctly. Nevertheless, the advantages are very useful, and not only are simulators regularly used in computer vision and robotics tasks [13], but they have also become an effective method for making progress in the development of self-driving cars [14]–[16]. To train autonomous vehicles, commercial and open-source computer games have been used in the past [17]–[19]. However, since they are not developed for self-driving research, they are not flexible enough to generate a wide variety of different scenarios.

One fundamental problem of the self-driving car research at the moment is that the results of different companies and teams are not easily comparable because the performance is often measured in miles per disengagement or similar metrics [20]. These statistics neglect the fact that different conditions greatly vary the difficulty of autonomous driving. For example, driving many miles without disengagement on a highway is way simpler than driving the same amount of miles through a city with dense traffic. To address this problem, CARLA also features a leaderboard [21]. This allows different teams to have a fair competition against each other and to be able to compare the research progress. Every research team can upload an agent

to the leaderboard server, where its performance is tested in an unknown city with stochastically placed events along the routes. To make the comparison as fair as possible, only a restricted set of sensors is allowed in these challenges. Each evaluation is then assigned an average driving score over N routes. This score is calculated by Equation 2.1. For the i-th route, the score depends on the percentage of the route distance, completed by the agent (R_i) and an infraction penalty to punish the agent's errors.

$$\text{driving score} = \frac{1}{N}\sum_{i}^{N} R_i P_i \tag{2.1}$$

Each infraction type has a unique penalty coefficient p between 0.5 and 0.8, the penalty P_i for the i-th route is defined by the product of all infractions that occurred along the route (see Equation 2.2). If there are no infractions, P_i is set to 1.

$$P_i = \prod_{j}^{\text{infraction type}} \left(p_i^j\right)^{\#\,\text{infractions}_j} \tag{2.2}$$

To reduce the variance of the simulation even further, each route is evaluated several times and the average score is calculated. This leads to more comparable results between different algorithms and techniques, which is necessary to advance research in the field of self-driving vehicles.

CARLA version 0.9.10.1 is used in all experiments performed for this thesis.

2.3 Imitation Learning

Humans, as well as animals, are able to acquire new behaviors by observing and imitating others. However, this so-called imitative learning not only helps living creatures to learn. For some time now, a similar concept has also been successfully used in the field of machine learning and especially in the development of autonomous vehicles. Imitation learning (IL) is a supervised learning approach, in which an agent is learning by imitating the behavior of an expert.

ALVINN [22], developed in 1988, is one of the earliest self-driving car experiments. It was trained by behavior cloning, the simplest form of imitation learning. Its architecture consists of a 3-layer neural network, that takes images from a camera as input and produces a vector with the desired direction as output by learning from a driver who steers the vehicle during training. Today, behavior cloning is a widely used and promising technique to train autonomous vehicles [23], [24]. In behavior cloning, an expert policy π^* is first rolled out in the environment to collect training data and later the agent policy is trained based on this dataset.

One immediate drawback of this supervised approach is that if the expert is not perfect, the agent will learn the mistakes the expert did when collecting the training data and therefore it will most likely not be able to outperform the expert, since it is only trying to imitate the expert's behavior. Therefore the expert's performance can be seen as the upper bound for the agent's performance. As the agent is trained on data from rollouts of the expert policy in an environment and not directly from the policy itself, there may be situations that are not in the training dataset and therefore the agent behaves differently than the expert would do in these situations. It is therefore important to have a diverse dataset that covers as many different situations as possible. Even if an agent only has a very small error compared to an expert, this error can accumulate over time and therefore the agent might end up in an unseen situation. An algorithm called DAgger [25] can be used to address this issue. In this work, this problem is decreased by using a sufficiently large dataset (section 4.1.2), generated by an imperfect agent, that ends up being in a wide variety of states. This is made possible due to data collection being very cheap in a simulator and the variety of situations being smaller compared to the real world.

In general, an end-to-end driving agent can be trained with the sensory input of many different sensors like RGB cameras, radar, and LiDAR sensors. Current state-of-the-art methods often use a combination of the aforementioned sensors and there is much research on how to fuse those inputs [26]. This work focuses on training agents on RGB camera images from one front-facing camera without additional LiDAR or radar sensors.

In everyday road traffic, there are frequently ambiguous situations like intersections where the desired action not only depends on the current perception but also on the high-level route. For these situations, the agent needs additional information about the high-level route plan to be able to behave as desired. This is where conditional imitation learning (CIL) [16] is used as the control network not only takes the current observation of the scene as input, but also an additional high-level command. In practice, this command could for example be given by a GPS navigation system. Codevilla et al. [27] further developed CILRS, an improved version of CIL and, although they trained on a large dataset, they observe large variance from training initialization and sampling order. Furthermore, the agent shows problems in generalization as its performance drops significantly in new environments and new weather conditions. This is addressed by Behl et al. [24], who show that this variance can be decreased if semantic segmentation images are used as perceptual input to the model instead of raw color images. This additional intermediate representation also allows for easier transferring of the driving policy in between different environments [28].

2.4 Semantic Segmentation

Semantic segmentation is the task of assigning a class label to each pixel of an image. It is commonly used in computer vision tasks to better understand the shape, category, and size of objects in an image. For example, converting an RGB image of a street scene into a semantic segmentation image can greatly reduce the complexity of the image but still encode relevant information like where the road, line markings, and other traffic participants are. Therefore semantic segmentation can be especially useful for autonomous vehicles to better understand their surrounding environment. Examples of different images of street scenes and their respective label maps can be seen in Figure 2.3.

Semantic segmentation combines the tasks of object detection, localization, and the classification of each object. While there are a few algorithms to segment images in areas of interest, the rise in popularity of deep convolutional neural networks pushed the state-of-the-art performance significantly. Fully convolutional networks (FCN) [29] are one of the earliest proposals using deep learning for semantic segmentation that achieve state-of-the-art performance. Today, a wide variety of architectures exist [30].

While the semantic labeling of images is a very time-consuming task, there are commonly used datasets [31]–[33] which are specifically designed to improve understanding of urban street scenes and accelerate self-driving research. Among other information, they contain images with corresponding semantic segmentation labels. Those datasets are also used for benchmarks, which enables the comparison of different models. For easier implementation, there are open source libraries [34] that include pre-trained models. This can significantly speed up the training process. Recent studies show that when autonomous vehicles are trained via behavior cloning on semantically segmented images, there is no significant drop in performance when the segmentation itself is trained on a relatively small dataset of only a few hundred annotated images [24].

As CARLA provides ground truth segmentation images, labeling of images is not an issue. However, in the official leaderboard, those ground truth segmentations are not allowed to be used. CARLA's segmentation camera classifies each pixel into one of 23 different classes[4]. Nevertheless, a few additional classes are necessary, because these provided classes do not encode all essential information. As CARLA provides only one class for traffic lights, independent of their current state, separate classes for 'red', 'yellow', and 'green' are added (see Figure 2.3). An additional 'stop sign' class is created since, in CARLA's semantic label maps, all street signs are of type 'traffic sign'. Lastly, an additional class for stop lines is included, as many intersections in CARLA are marked by stop lines instead of stop signs. Overall this modified segmentation camera classifies all pixels into one of 28 different classes.

[4] https://carla.readthedocs.io/en/0.9.10/ref_sensors/#semantic-segmentation-camera

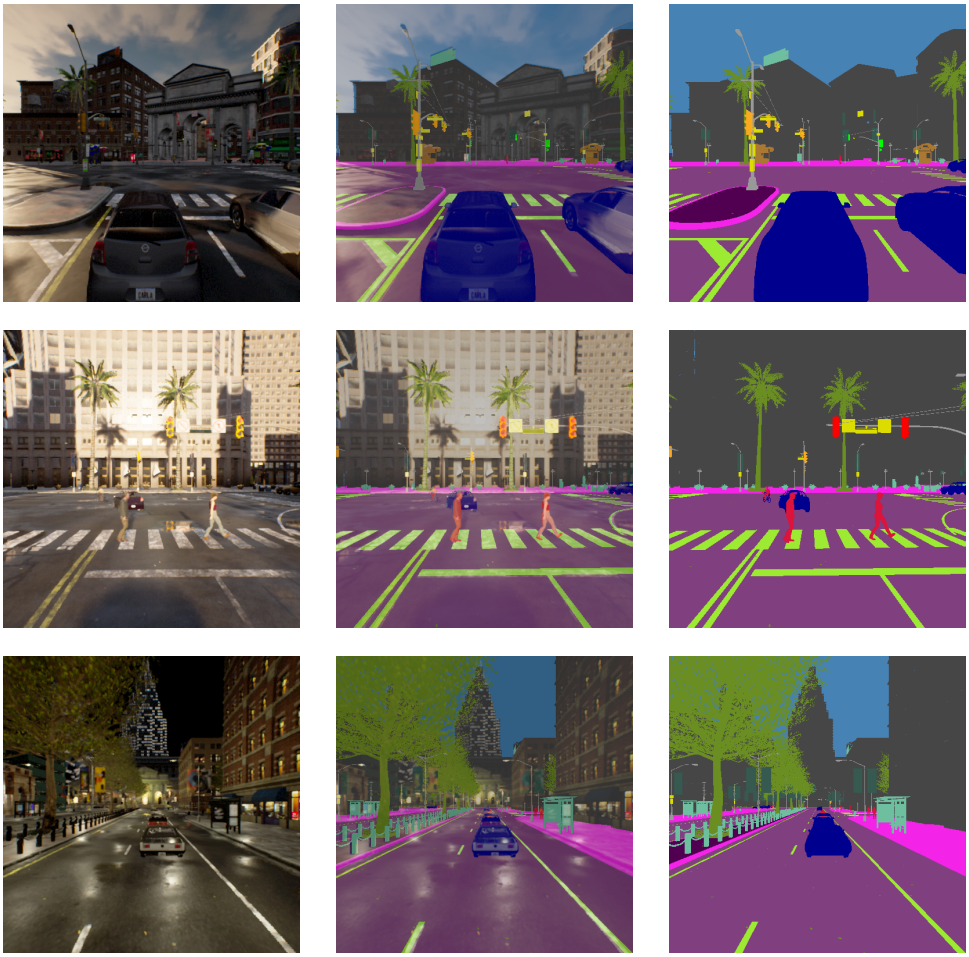


Figure 2.3: Color images (left) with respective semantic label maps (right) from the CARLA simulator. For the purpose of presentation, the semantic label maps are converted to color images. Different colors correspond to different classes (see Figure 4.1 for legend). In the center column, the camera view and the semantic label map are overlayed. The top two rows include the additional classes for red and green traffic lights that are not part of CARLA's segmentation camera.

3 Methods

This chapter deals with the implementation details of the algorithms that are relevant to this thesis. In section 3.1, the research questions and objectives of the work are outlined. Section 3.2 explains how semantic segmentation is used as visual abstraction and further presents the segmentation model's architecture. An introduction of the waypoint model and the random color model follows in section 3.3. The waypoint model is the architecture that is used for all studies conducted in this work. The random color model is a modification of the waypoint model which can be used in combination with genetic algorithms (section 3.4) to find relevant classes in semantic label maps. Finally, in section 3.5, another abstraction method used in the experiments, called affordances, is presented.

3.1 Research Questions

The research objective of this work is to improve the understanding of the influence of visual abstraction on the driving performance of autonomous vehicles that are trained by imitation learning. To get a deeper insight into this problem, the following questions serve as a framework for this study:

- **Do visual abstractions in the form of semantic label maps ease the imitation learning process?**
  Previous research suggested that using semantic label maps as an intermediate representation for an end-to-end trained driving agent might not only improve the agent's performance but also lower the variance due to different training seeds [24]. These experiments were run in CARLA version 0.8.4, which is simpler than the current version. Thus the first question is whether those results can be confirmed in the newer version with more complex scenarios, environments, and weather conditions and, whether those results are model specific or whether they can be applied to different model architectures. Further, the more complex simulation might lead to a shift in importance for the different classes that are hypothesized to be relevant.
- **What classes are relevant for autonomous driving?**
  Choosing the right classes for the semantic representation is key. Some classes are relevant for the driving task and other classes are not as relevant or even obstructive. Fewer classes not only mean less labeling effort every time a new dataset has to be created, but it might even improve the performance of a driving

agent. The additional information that differentiates whether there is a wall or a building beside the road probably should not influence the driving agent's decisions. However, this unnecessary additional information could cause the neural network to make an incorrect decision. Hand designing a working subset of classes is possible but there is no guarantee that all relevant information is encoded that way. There could be complex relationships between different classes and an automatically trained agent could get valuable information from classes that are not obvious to a human. Furthermore, there might be classes that a person considers relevant, but which do not encode important information. Ideally, the relevant classes are learned automatically so that all relevant information will be encoded.

- **What metrics and hyperparameters correlate with the agent's performance?** When training a model, the selection of a correct loss function is vital as it defines the direction in which the training should converge. However, for behavior cloning with a given dataset, this loss does not necessarily correlate well with the driving score the agent is able to achieve in the simulator, as there is no way to infer this information in an offline manner. If visual abstraction is used, the driving task should become easier as irrelevant information is abstracted away. Does this influence the model choice and choice of hyperparameters?

- **Can additional affordances further improve the performance?** Some attributes of the environment may be better encoded in a single value instead of embedding them into a semantic label map. For example, it may be easier to train a separate model to detect whether the driving agent has to stop for a red traffic light than to have a separate class for red traffic lights in the semantic segmentation model. Furthermore, a simple rule-based controller would be able to infer an action from an affordance. When the same information is encoded in an image, this task is more difficult. Such additional parameters could improve the model's robustness and could be an essential factor in a safety-critical application as self-driving cars. Are there any affordances that can improve a model's performance and can carefully selected additional parameters also decrease the performance?

3.2 Visual Abstraction

Visual abstraction, as it is used in this work in the context of self-driving vehicles, describes the process of abstracting and simplifying recorded camera images in a meaningful way, such that it is easier for an end-to-end model to perform the task of self-driving. This so-called intermediate representation is designed such that unnecessary details are removed but the essential information for the task is obtained. This can not only simplify the training process but can also play an important role

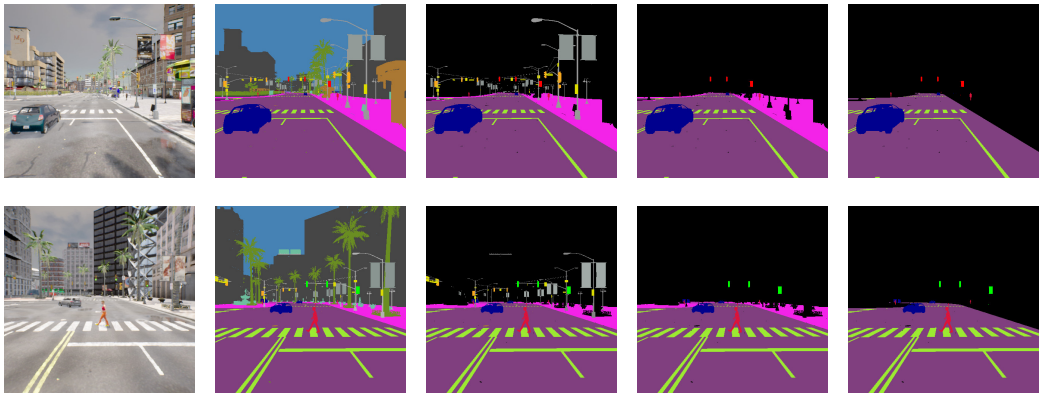


Figure 3.1: Visualization of different levels of visual abstraction with semantic label maps. The RGB image is transformed into a semantic representation and subsequently different classes are removed from the images. The rightmost semantic label map only contains the classes: 'road', 'vehicle', 'pedestrian', 'line markings', 'red light' and 'green light'.

when a model is transferred from a simulator to the real world[1].

Behl et al. [24] introduced semantic label maps as the intermediate representation for an end-to-end trained CILRS model. They show that this addition can improve the agent's performance. Subsequently, they simplify the representation even further by handpicking a subset of important classes, which increases the success rate even further. They hypothesize seven classes to be most relevant: 'road', 'sidewalk', 'vehicle', 'pedestrian', 'line markings', 'red light' and 'green light' and later show that the additional 'sidewalks' class might not help the agent either. For training of the self-driving agent, it is possible to directly get ground truth semantic label maps from the CARLA simulator. Without a simulator, fewer classes also mean less labeling effort as all unselected classes can be mapped to one 'unlabeled' class. This is an additional benefit. Figure 3.1 shows different levels of visual abstraction for multiple street environments in CARLA. The two rightmost semantic label maps only contain the classes that are hypothesized by Behl et al [24].

Semantic label maps are well suited as an intermediate representation because if they are converted to RGB images, humans are able to easily interpret them. This can improve the interpretability of the autonomous agent as a whole which is vital for a safety-critical system like self-driving cars.

One goal of this work is to find the smallest possible subset of relevant classes. To reduce the number of classes, every class in the semantic representation could be mapped to every other class. This class mapping is defined by a vector of length 28. The i-th entry c_i of this vector defines the new class, to which all pixels of the i-th

[1] `https://youtu.be/htdsPSgbLQo?t=1227`

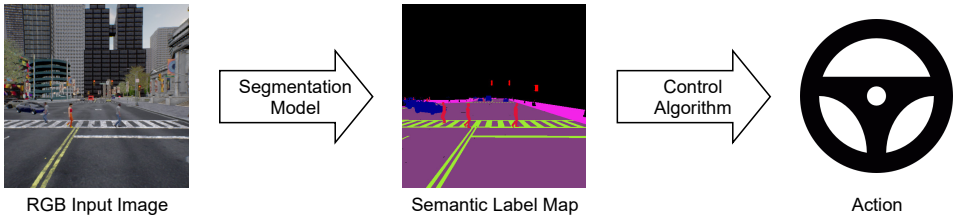


Figure 3.2: High level overview of the modular approach to incorporate semantic label maps as intermediate representation.

class are mapped to. Therefore this goal can be rephrased to the search of suitable class mappings. To prevent different class mappings from mapping to the same set of output classes, the restrictions in Equation 3.1 are defined.

$$\begin{aligned} c_0 &= 0 \\ c_{i+1} &\leq \max\{c_0, ..., c_i\} + 1 \end{aligned} \tag{3.1}$$

There are different methods to directly or indirectly make use of such visual abstractions [24], [35], [36]. In this work, only the modular approach is used (see Figure 3.2). It is the simplest and most straightforward method. In the first step, the camera image is transformed into an intermediate representation. This representation is then the only input to the control algorithm. This approach is easy to implement and has been shown to improve the performance of the model. Another great benefit is that the intermediate representation is human readable.

When the self-driving agent is deployed to the CARLA leaderboard or real world, it does not have access to the ground truth semantic label maps from the simulator. To get semantic label maps from RGB camera images, the following segmentation model is trained.

3.2.1 Segmentation Model

The segmentation network maps the RGB input images to one-hot encoded semantic representations. It is the first step in the agent's driving pipeline. The model is a FCN [37] utilizing a ResNet-50 [38] backbone, pre-trained on the cityscapes dataset [32]. The architecture, hyperparameters and pre-trained model weights are taken from MMSegmentation [34].

The weights are fine-tuned on the CARLA dataset (section 4.1.2). Therefore, custom class weights w_j are used because the individual classes j in a typical street scene are very unevenly distributed. Using p_j as the number of pixels of class j for all N classes in the dataset, individual weights w_j are calculated by Equation 3.2.

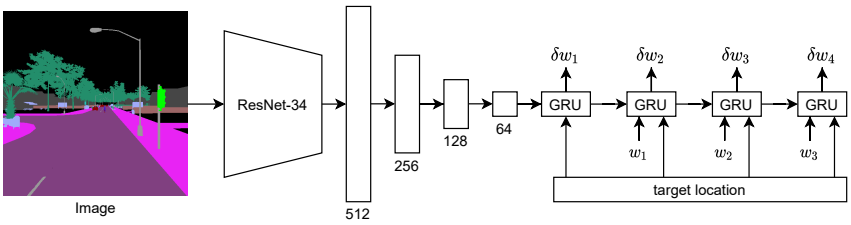


Figure 3.3: Overview of the waypoint model's architecture. The first step is a ResNet-34 followed by linear layers to encode the semantic label map into a vector of length 64. Along with a higher level target location, this is fed into a gated recurrent unit that generates waypoints for the model to follow.

$$w_j = 1 - \frac{p_j}{\sum_{i=0}^{N} p_i} \tag{3.2}$$

3.3 Waypoint Model

The driving agent uses the architecture seen in Figure 3.3. This model is able to predict future waypoints based on an input image and a target location. The architecture is based on the Auto-regressive IMage-based waypoint prediction (AIM), as this has proven to be a much stronger model for autonomous driving than LBC and CILRS, the previous state-of-the-art for imitation learning in autonomous driving [26].

Input to the model is a 256×256 one-hot-encoded semantic representation of the current scene. During training and for the privileged analysis, this image comes directly from CARLA's segmentation camera. During deployment, it comes from an upstream semantic segmentation network (section 3.2.1). Gharaee et al. [39] show that an agent can be more successful when trained on ground truth segmentation images even when they use inferred segmentation images during deployment. As in AIM, the image is cropped from a 400×300 front-facing camera to remove the distorted edges of the image.

The first component of the waypoint model is a pre-trained ResNet-34 [38] architecture to encode all the relevant information as a 512-dimensional feature vector. The number of input channels of the first ResNet layer varies in size as the number of classes in the input image changes. The following three linear layers further reduce the size of this feature vector to 64.

Subsequently to the encoder, this vector is fed into a single-layer gated recurrent unit (GRU) [40] cell with 64 hidden neurons. This cell is run for $T = 4$ timesteps, each 0.5 seconds apart from each other, to produce four future waypoints as output. In each timestep, the GRU cell takes coordinates of a target location as well as the

previous waypoint w_{i-1} as input. After each timestep, a single linear layer that takes the 64 hidden neurons of the GRU cell as input, produces new relative waypoint coordinates δw_i. In this context, the target location and the waypoints are relative to the position of the vehicle. For the prediction of the first waypoint w_0 is set to the position of the vehicle $(0,0)$. These target locations are high-level GPS coordinates provided by CARLA. In contrast to the output waypoints, they can be very far from each other and are therefore not sufficient for autonomous driving. The target locations encode the overall route plan for the network. The next three waypoint predictions are then calculated by $w_n = w_{n-1} + \delta w_n$.

The model's policy $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ is trained by a supervised learning algorithm called behavior cloning. That is for a given state distribution P^* that is provided by an expert, the goal is to find policy parameters θ that satisfy Equation 3.3. The state distribution contains the states seen by the expert s^* and the actions the expert took in those situations a^*. Depending on the experiment, L_1 or L_2 distance between the expert waypoints and the predicted waypoints are used as loss function $\mathcal{L}$. A deeper investigation into the effect of the loss function can be seen in section 4.4.1.

$$\underset{\theta}{\operatorname{argmin}} \mathbb{E}_{(s^*,a^*)\sim P^*} \left[\mathcal{L}(a^*, \pi_\theta(s^*)) \right] \tag{3.3}$$

CARLA expects low-level steering, throttle and braking commands to control the vehicle. During deployment two PID controllers are used to calculate the desired throttle and steering commands based on the predicted waypoints, generated by the waypoint model. For a given error value e, the control signal u is computed by:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt} \tag{3.4}$$

3.3.1 Random Color Model

Using the waypoint model it is possible to test and compare the performance of driving agents trained using semantic label maps with different classes. The problem is that generating the driving score for one such class mapping includes training a new agent with the corresponding class mapping and subsequently evaluating it. This can take up to two days. Hence, both, training and evaluation, are simplified to shorten the run time for a single class mapping significantly.

To be able to test many different class mappings in a shorter amount of time, the waypoint model has to be modified in such a way that it can process segmentation label maps for any class mapping. As the class mappings change, the number of elements in the one-hot semantic label maps changes with the number of classes, as they are encoded by the individual one-hot positions.

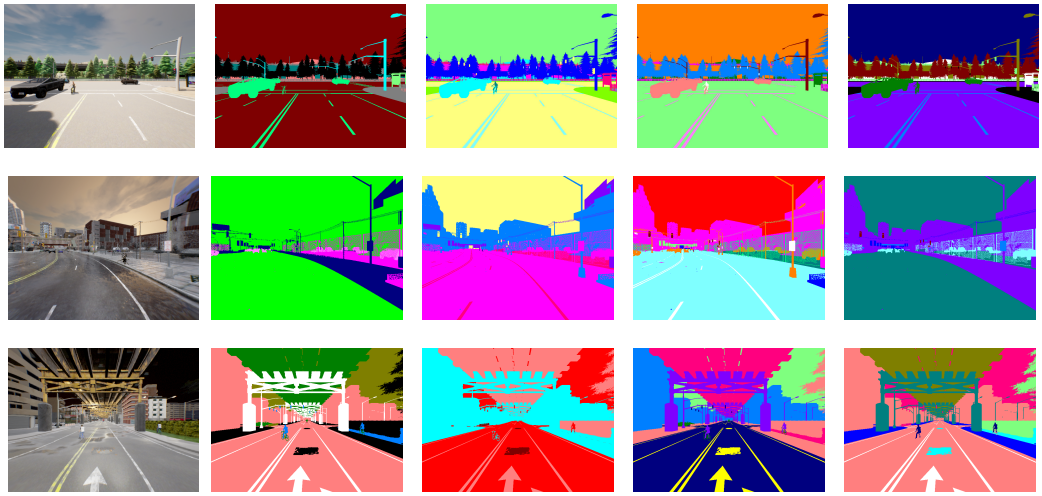


Figure 3.4: RGB images (left) of street scenes and four different corresponding random color images. The random color images are created by randomizing class mappings and subsequently assigning random colors to each class.

Therefore the Random Color Model (RCM), a generalized version of the waypoint model, is introduced. The RCM is based on the waypoint model with one additional module in the encoder. This gives it the ability to process one-hot segmentation label maps produced by different class mappings in the segmentation process. In this module, each semantic label map is mapped to a random color image by mapping each pixel to a unique class color, which is randomly chosen from 27 numerically equidistant colors (see Figure 3.4). To achieve as much variety as possible, this color selection is done again every time an image is loaded. The algorithm transforms the semantic representations into RGB images, in which the individual colors do not encode any information. Therefore it can be used to analyze complex relationships between the different classes.

The RCM can simply be trained once with the same dataset as the original waypoint model. For the model to learn to generalize to different class mappings, it is trained using random color images. Those random color images are generated during training from the semantic label maps in the dataset in a three-step process: The first step is to randomly choose the number of classes $c \in [2, 11]$. Eleven is chosen as the upper limit, as this number is sufficiently above the seven classes identified to be relevant by Behl et al. [24]. The second step is to generate a new random class mapping using c classes and to map all the classes in the semantic representation to the new classes. In the third step, this new semantic label map is transformed into a random color image using the same algorithm as during inference.

Even in the simulator, evaluations are very time-consuming. Therefore, evaluating the RCM for each class mapping in CARLA is infeasible and thus the RCM's loss over

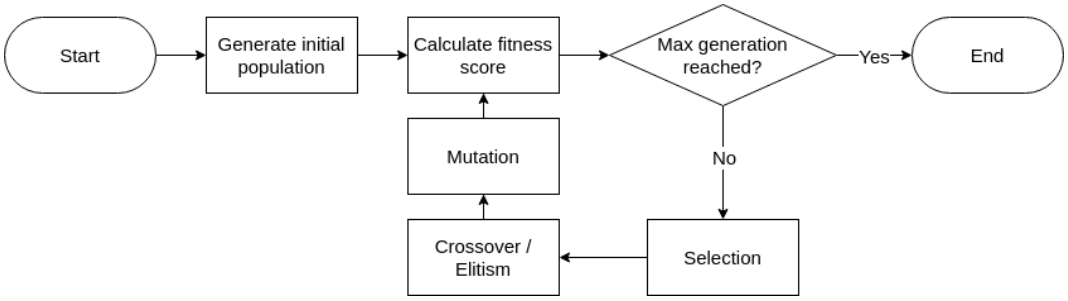


Figure 3.5: Overview of the implemented genetic algorithm's architecture. The algorithm is run until a predefined number of generations is reached.

the validation set is used as an estimate for the driving score. This reduction comes at a cost because although correlated, a lower validation loss does not necessarily mean that the agent's performance is better. The correlation between loss and performance is discussed in more detail in section 4.6.

Using the RCM allows to quickly filter out promising class mappings.

3.4 Genetic Algorithm

As stated above, the goal is to find class mappings that reduce the number of classes in the semantic label maps. The modified segmentation camera of CARLA returns one of 28 classes for every pixel in an image. Even in this simple simulation case, this leads to more than 6×10^{21} [41] possible unique assignments, which makes it infeasible to simply brute force through all possible combinations. The RCM introduces a metric for all possible class mappings, so the search for a suitable mapping can be reframed as an optimization problem. This can be solved by genetic algorithms (GAs).

First introduced by Holland in 1975 [42] GAs became a popular tool to solve nonlinear optimization problems. Today they are used in a wide variety of applications [43].

Inspired by biological evolution and natural selection, the GA iteratively selects and combines the best performing individuals from a population of potential solutions to produce offspring for a new generation (see Figure 3.5). Through this process, solutions with desired properties are kept in the population and these properties are amplified over time.

The GA used in this work is defined by the following parameters:

- Population:
  The set of all possible solutions is called population. The individual solution in such a population is called a chromosome. The single entries of the chromosomes are called genes. Therefore the class mappings defined in section 3.2

can directly be used as the population, where the n-th gene defines the new class, to which all pixels of class n are mapped. The population is initialized randomly in such a way, that it is uniformly distributed over the number of classes to which the chromosomes map.

- Fitness function:
  The fitness function is a function that assigns a performance score to each of the chromosomes. The higher the score, the better the chromosome solves the optimization problem. Chromosomes with a higher fitness score are more likely to be chosen for crossover to produce offspring in the new generation. This score is set to the inverse of the loss, a given chromosome achieves in the RCM.

- Crossover:
  In the crossover step, two chromosomes are mixed to produce offspring that are added to the next generation. This is the most crucial step of the GA and the crossover strategy needs to preserve information from the parent chromosomes for the next generation and at the same time inherit information from both parents to both children. At first, two parents are selected by random choice. The higher the fitness score of a chromosome, the more likely it is to be selected as a parent. The selection of the crossover operator changes the performance and exploration of the GA [44]. As the chromosomes represent class mappings, it is not possible to use simple mixing strategies like single-point crossover to produce offspring. Therefore algorithm 1 is proposed as a new recombination strategy. In each step, all genes with the same value are copied from a parent to the respective child. This way correlations between the classes are maintained.

- Mutation:
  In the mutation step, a few randomly selected genes are modified such that the chromosome is slightly mutated. This is used to maintain genetic diversity within the population. It is important to balance the probability of the mutation correctly, as too much mutation leads to poor performance. Furthermore, the mutation must only produce valid solutions to the optimization problem. The class mappings can be mutated by flipping single genes to new values.

- Elitism:
  Elitism is a process in which the best chromosomes of a population are directly copied to the next generation. Introduced by De Jong [45], this helps the algorithm to transfer the most important information into the next generation and guarantees that the best performing chromosomes are not lost in upcoming generations.

Algorithm 1: Crossover Strategy for Class Mappings

Input: p_1, p_2, Parent chromosomes of length n; j # classes
Initialize c_1, c_2 as empty arrays of length n
$S \leftarrow$ *shuffle* ($[0 \ldots (j-1)]$);
for k *in* S **do**
    for $i \leftarrow 1$ *to* n **do**
        if $p_1[i] == k$ *and* $c_1[i]$ *is None* **then**
            $c_1[i] \leftarrow k$;
        end if
        if $p_2[i] == k$ *and* $c_2[i]$ *is None* **then**
            $c_2[i] \leftarrow k$;
        end if
    end for
    swap (p_1, p_2);
end for
for $i \leftarrow 1$ *to* n **do**
    if $c_1[i]$ *is None* **then**
        $c_1[i] \leftarrow p_1[i]$;
    end if
    if $c_2[i]$ *is None* **then**
        $c_2[i] \leftarrow p_2[i]$;
    end if
end for
return c_1, c_2

3.5 Affordances

Another hybrid driving paradigm between modular pipeline and end-to-end learning is the direct perception (DP) [19] approach. In DP a manually designed low-level intermediate representation, so-called affordances, is used to infer the vehicle's actions. Sauer et al. [46] improved this model and have shown that such a low-level affordance representation can provide the necessary information for autonomous driving in the CARLA simulator. For their conditional affordance model (CAL) they trained a neural network to infer the affordances directly from the RGB camera images and designed a rule-based controller to output low-level commands based on those affordances.

The expert agent uses affordances and a rule-based controller to drive inside the simulation as well. In the expert's case, those affordances are based on ground truth information directly obtained by CARLA. They determine whether the agent must stop. In addition to the affordances, the expert controller has access to ground waypoints to determine the correct action. More information about the expert agent can be found in section 4.1.1.

Similar to the modular pipeline approach, the benefit of such an intermediate representation is that these affordances are easy to interpret. Furthermore, they can be complemented by rule-based algorithms, which could increase the overall robustness and interpretability of the driving agent. However, like in the modular pipeline approach, it is possible for the affordances to be a suboptimal encoding and to not encode all the required information. Additionally, a more complex driving task may require more and different affordances, as there may be more information that has to be encoded.

In this work, a similar full affordance model is used as a baseline. Further details on the implementation and training of this model are described in section 4.1.5.

The waypoint model can also make use of such affordances. A separate model that is specifically trained on a few affordances might outperform the waypoint model in the ability to predict those affordances. Therefore, an additional affordance model is trained on the three parameters red light, vehicle, and pedestrian. There are multiple ways to add those additional parameters to the waypoint model. The simplest approach is to use them as emergency stop parameters even after the PID controller generates the low-level commands, as those three parameters are all stopping conditions for the expert. If one of the affordances is set to true, a simple rule-based controller immediately stops the vehicle. If all affordances are false, they are ignored. This architecture has the great advantage that the correct action is guaranteed to be executed as soon as the obstacle is detected by the affordance model. If, on the other hand, the affordance parameter is included in the GRU cell which generates waypoints, from which the action is inferred, this is not guaranteed. As such affordance models can also be designed in a way that it outputs its confidence, a driving agent with this architecture can be fine-tuned after training to get a more

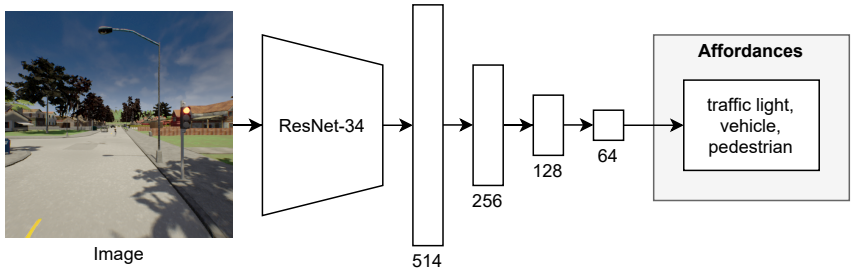


Figure 3.6: Architecture of the Affordance model.

passive or aggressive driving style by changing the threshold for the detection of some affordances.

The affordance network is identical to the encoder part of the waypoint model (see Figure 3.6) with one additional linear layer with three neurons and a sigmoid activation function. For training, a 'binary cross-entropy loss' with individual weights for positive samples is used. For affordance i and number of positive (P_i) and negative (N_i) samples in the dataset, the weight w_i for each affordance is defined by $w_i = \sqrt{N_i/P_i}$.

4 Experimental Evaluation

This section contains implementation details and results for the conducted experiments. Section 4.1 provides an overview over details of the dataset, the evaluation setting as well as the baseline models. Following in section 4.2, preliminary experiments show that visual abstraction can improve an agent's performance. Section 4.3 contains the proof of concept for the extraction of relevant classes using the random color model in combination with the genetic algorithm as well as the set of classes, proposed by this method. In parallel to the learning of the classes, the experiments in section 4.4 are carried out to get a deeper insight into important hyperparameters as well as the relevant classes. In section 4.5, the influence of additional affordances on the driving performance is investigated. Section 4.6 contains a statistical evaluation of all preceding experiments. Finally, the last two sections contain experiments on the influence of the segmentation model on the performance of the visual abstraction (section 4.7), as well as a study examining the impact of visual abstraction on other model architectures (section 4.8).

4.1 Implementation Details

The training and validation datasets are created by a rollout of the expert agent. In this section, this process is described in detail. Furthermore, the experimental setup, implementation details, and hyperparameters for the waypoint model as well as the random color model are presented.

4.1.1 Expert Agent

For behavior cloning an expert policy π^* is necessary. For this work, the expert is a rule-based agent, that has access to privileged information from the simulator. The privileged information includes a target point near the vehicle, four brake criteria, and one of two target speeds. The target speeds are 7.0m/s for straight driving and 4.0m/s in all turns. The four braking criteria are 'red traffic light', 'stop sign', and whether the ego vehicle is on a colliding path with pedestrians and other vehicles. If at least one of those parameters is true, the agent stops. Given the next target point and a target speed, two PID controllers determine throttle and steering for the expert.

The expert agent used for this work has problems with its rule-based stop sign behavior, therefore it is not expected for the models trained on this expert policy to show correct behavior in these situations. In some situations, this leads to a reduced driving score for those models.

As many parameters of the expert are manually tuned, it does not achieve a perfect driving score of 100 on every route, either. On the 42 evaluation routes described in section 4.1.3, the expert achieves a driving score of 61.37. This defines the upper bound for the performance of the models trained on the expert policy.

4.1.2 Dataset

The training dataset consists of 142751 training samples and 51454 validation samples. Each sample contains an RGB camera image with a corresponding semantic label map, as well as the current position, velocity, and affordances. The cameras have a field of view of 100° and are mounted 1.3m in front and 2.3m above the vehicle's center. All images are saved at a resolution of 400×300. The dataset is collected by running the expert policy and saving all necessary information at a rate of two FPS. The positions of consecutive frames are used as waypoints for training.

As described in section 2.4, five additional classes are added to the original classes from CARLA. Figure 4.1 shows the distribution of the individual classes in the dataset's semantic label maps. The uneven distribution must be taken into account when training the segmentation network.

All eight towns available in CARLA 0.9.10 are used to collect the dataset. The data is collected on the training and test routes that are provided in the CARLA leaderboard repository[1] as well as additional routes since the provided sets contain relatively few turns and intersections.

During collection, the weather condition changes for each frame to get more diversity in the dataset: It contains six different daytimes (Night, Twilight, Dawn, Sunset, Morning, Noon), each with its own visual challenges. In the dataset, there are seven different weather conditions (Clear, Cloudy, Wet, MidRain, WetCloudy, HardRain, SoftRain). Every time the weather changes, one of eight equally distributed sun azimuth angles is randomly sampled. This makes for a wide variety of different conditions. The official leaderboard additionally contains foggy weather conditions, these were however not applicable due to a current bug in CARLA which prevented these conditions on the machines used.

Furthermore, during data acquisition, adversarial scenarios are spawned at specified locations along the route. The scenarios are defined by adding further scenarios to the

[1] `https://github.com/carla-simulator/leaderboard/tree/stable/data`

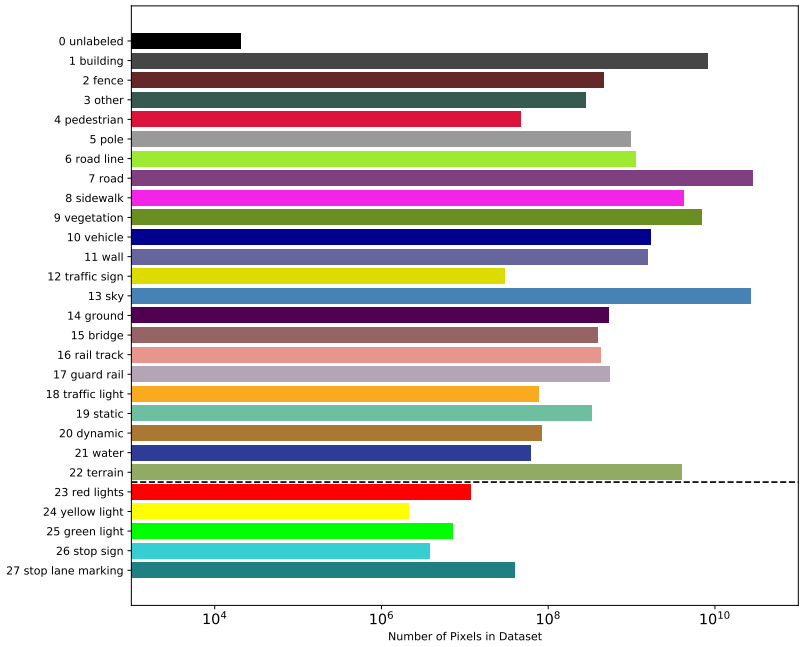


Figure 4.1: Distribution of pixels per class in the dataset plotted on a log scale. The number of pixels per class varies a lot in semantic label maps of street scenes. Classes that are encoded with fewer pixels are not necessarily less important. The dashed line separates the classes provided by CARLA (top) from those added for this work (bottom).

existing definitions provided in the CARLA leaderboard repository[2]. They contain sudden control loss of the ego vehicle, pedestrians and bicyclists unexpectedly crossing the street as the ego-vehicle approaches, as well as other vehicles running red traffic lights. Moreover, at intersections, the ego-vehicle has to make unprotected left turns with oncoming traffic and right turns with crossing traffic. The traffic scenario types are provided by CARLA and are selected from the NHTSA pre-crash topology.

[2] `https://github.com/carla-simulator/leaderboard/blob/stable/data/all_towns_traffic_scenarios_public.json`

4.1.3 Evaluation Routes

As there are only limited resources available to evaluate a model on the official leaderboard server, the agents are evaluated locally with the following settings: to get comparability between the different agents, all models are evaluated on the same set of 14 predefined routes through Town01 to Town06. The routes' lengths range from 110m to 2811m and they contain wide of a variety of different situations and tasks for the vehicles to fulfill. To reduce the variance of the evaluations, each route is evaluated three times with a different weather condition and time of day, so overall 42 routes are used for evaluation.

As in the data collection step, during evaluation adversarial scenarios are spawned along the routes. Additionally, some agents are evaluated a second time without any adversarial scenarios enabled.

4.1.4 Waypoint Model Implementation

The waypoint model takes on-hot encoded semantic label maps of size 256×256 as input. The semantic label maps contain a separate position for the unlabeled class. The input images are cropped from camera images to remove distorted regions. The two-dimensional target location coordinates are relative to the ego-vehicle.

In this work, the model is implemented with PyTorch [47]. PyTorch features model weights for ResNets that are pre-trained on the imageNet dataset. To accelerate the training process, those pre-trained models are used for the encoder's ResNets.

The models are trained for 100 epochs with a batch size of 192. For training, an Adam optimizer is used with a constant learning rate of 0.0001.

To generate steering and throttling commands from the predicted waypoints, two separate PID controllers are used. Their hyperparameters are taken from the LBC [35] code base[3] provided by the authors. The turn controller uses $K_P = 1.25, K_I = 0.75, K_D = 0.3$, the speed controller $K_P = 5.0, K_I = 0.5, K_D = 1.0$. For both controllers, a finite time window of 40 time-steps is used to calculate the integral. The error is initialized with 0 for the whole window. The desired speed depends on the euclidean distance between the first two waypoints $(w_{1,t}, w_{2,t})$ at time step t. If the desired speed is either below 0.4m/s or below the current speed, the throttle is set to 0 and brake to 1. Otherwise, the throttle value is clipped at a maximum of 0.75. The desired steering angle depends on the angle between the current trajectory and the path towards the aim location $a_t = (w_{1,t} + w_{2,t})/2$. The steering command is clipped to a range between -1 and 1.

[3] `https://github.com/bradyz/2020_CARLA_challenge`

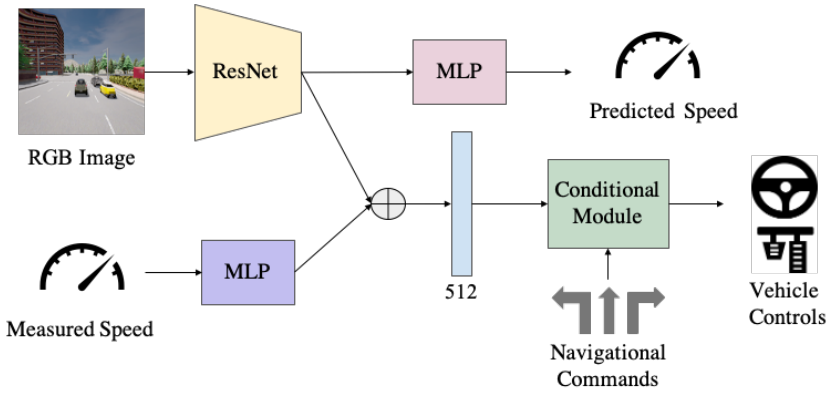


Figure 4.2: CILRS model architecture. (The Figure is taken from the TransFuser github repository[4])

4.1.5 Baselines

CILRS

CILRS [27] is a conditional imitation learning approach based on a ResNet-34 encoder. The input to this model is a single front-facing camera image, the measured speed, and an additional conditional command. The condition is a discrete high-level navigational command between 1 and 6, which encodes what the ego-vehicle is supposed to do in ambiguous situations. In addition to the four directional commands that are already used in the original paper, two commands (left lane change, right lane change) are added. The encoder is followed by a conditional module. This consists of six identical multilayer perceptrons, one for each command. They are implemented in parallel and at any time only the multilayer perceptron assigned to the current command is used. The conditional module outputs low-level vehicle controls (steering, throttle, and brake) that are directly used for driving. The network has an additional branch to predict the current speed, as this increases the model's performance significantly [27].

For the implementation used in this work, the code provided in the TransFuser codebase[5] is used.

[4] https://github.com/autonomousvision/transfuser/blob/main/cilrs/assets/model.png
[5] https://github.com/autonomousvision/transfuser

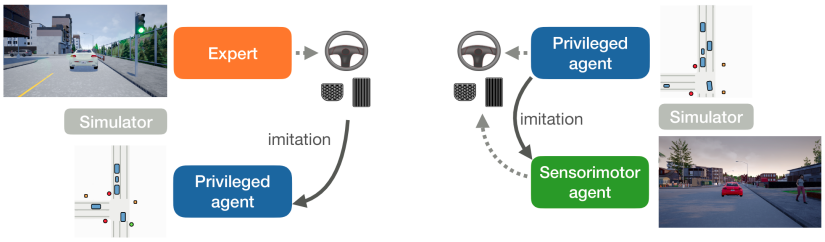


Figure 4.3: LBC architecture. (The Figure is taken from the LBC github repository[6]provided by the authors)

LBC

Learning by cheating (LBC) [35] is a supervised learning method in which path planning and perception are split into two separate training steps. In the first step, a privileged teacher agent is trained on birds-eye view semantic label maps. These semantic label maps contain all relevant information like other traffic participants and traffic lights and are easy for the teacher network to understand. Therefore it can 'focus' on path planning. Like the waypoint model, this teacher network outputs future waypoints. In a second step, the student network is trained under the supervision of the teacher network to predict the waypoints based on RGB images of three front-facing cameras. Therefore the target point is projected into the student's camera view. When deployed, only the student network is necessary to predict waypoints directly from the camera images and target point. Similar to the waypoint model, the student network's encoder is based on a ResNet-50. Through this two-step process, the student network makes indirect use of the teacher's privileged semantic label maps. This process is called distillation.

For the implementation used in this work, the code provided by the authors[7] is used.

Full Affordance Model

The full affordance model's architecture is inspired by the waypoint model. However, instead of predicting waypoints, the full affordance model predicts the affordances the expert uses when driving in CARLA. Figure 4.4 shows the full affordance model's architecture.

The model takes the image of a single front-facing camera, as well as a target location in ego-vehicle coordinates as input. The first linear layer after the ResNet-34 encoder has a size of 514, as the target location is concatenated to the 512-dimensional output vector of the ResNet. There are two linear layers in the last step: The first one only

[6] https://github.com/dotchen/LearningByCheating/blob/release-0.9.6/figs/fig1.png
[7] https://github.com/bradyz/2020_CARLA_challenge

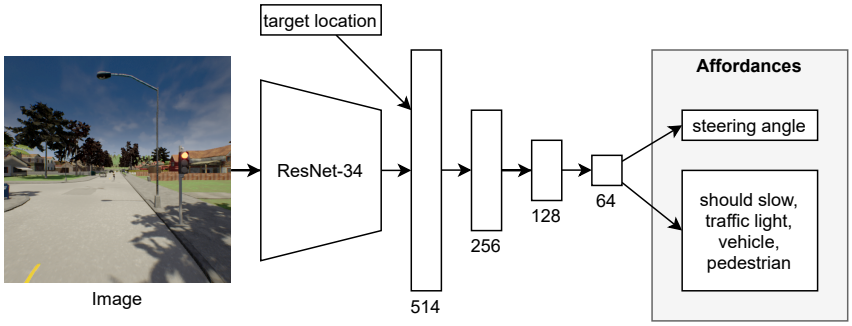


Figure 4.4: Full affordance model architecture.

has one neuron and predicts a steering angle between -1 and 1. Therefore it uses a hyperbolic tangent as the activation function and a L_2 loss. The second one has four neurons that are used to predict the boolean affordances used in the expert agent. The 'should slow' affordance is used by the expert to slow down in turns. Like the affordance model, this layer is trained with a weighted binary cross-entropy loss. The weights are calculated in the same way as for the affordance model in section 3.5. The overall loss is designed to be the sum of both losses.

4.2 Preliminary Experiments

Behl et al. [24] show that the performance of an agent trained on semantic label maps as visual abstraction instead of RGB images depends on the classes used in this map. The goal of the first experiment is to investigate whether their results can be transferred to the new version of CARLA. They conducted their experiments using a CILRS model in CARLA version 0.8.4, in which the available scenes are not as complex as in version 0.9.10. In the old version, there are fewer available ground truth segmentation classes. They hypothesized a subset of seven classes to be most relevant: 'road', 'sidewalk', 'line markings', 'vehicle', 'pedestrian', 'green traffic light', and 'red traffic light'. Furthermore, they show that the agent's success rate is higher when trained on semantic label maps with those seven classes than an agent trained on all available classes. They also show that excluding the 'sidewalk' class reduces the success rate in dense traffic and additionally removing the 'line marking' class significantly lowers the agent's performance.

The goal of the preliminary experiment is to test whether those results still hold in the new setting.

4.2.1 Setting

To answer this question the waypoint model is trained multiple times, each time with a different set of classes. For ease of implementation for the preliminary experiments,

a smaller ResNet-18 is used in the encoder. Subsequently, the models are evaluated on ten routes through Town05 using the ground truth semantic label maps from the simulator as input to the model. An L_2 loss averaged over the four waypoints is used during training. Inspired by Behl et al. [24] the following experiments are conducted:

- The baseline is an '*All Classes*' model, trained with all 28 available semantic classes.
- The 'Basis' subset consists of seven classes that are intuitively relevant for the driving task. It is similar to the seven class model of [24] with the addition of the stop class and consists of: 'road', 'road line', 'sidewalk', 'pedestrian', 'vehicle', 'red|yellow traffic light' and 'stop signs/line markings'. All pixels that do not belong to one of those classes are of class 'unlabeled'. To test the variance between different training seeds, a waypoint model with this class configuration is trained and evaluated three times.
- The '*No Line Markings*' set is identical to the 'Basis' set except for the removal of line markings. For removal, line markings are set to the road class as the agent would still be able to detect them if they had the class 'unlabeled'.
- By excluding the line markings and sidewalks from 'Basis', the '*5 Classes*' set is obtained.
- '*No Stop*' contains the classes 'road', 'road line', 'sidewalk', 'pedestrian', 'vehicle' and 'red|yellow traffic light'. Note: Stop points are a new feature in CARLA, so their influence on performance is not examined in the original paper [24].

4.2.2 Results

	Basis #1	Basis #2	Basis #3	All Classes	No Stop	No Line Markings	5 Classes
No scenarios	91	89	100	88	97	70	58
All scenarios	73	80	83	62	91	58	65
Overall	82	85	91	75	94	64	62

Table 4.1: Driving scores with and without adversarial scenarios for different agents on 10 test routes in Town05. Basis #1-#3 is the same agent trained with different training seeds. Due to variance in the training process and the fact that the evaluation in CARLA is not deterministic, the scores show some variance.

Overall the different agents achieve average driving scores between 62 and 94. In most cases, the score without adversarial scenarios is higher than the score obtained with all scenarios enabled. This driving score divergence is as high as 27 points for the 'All Classes' agent.

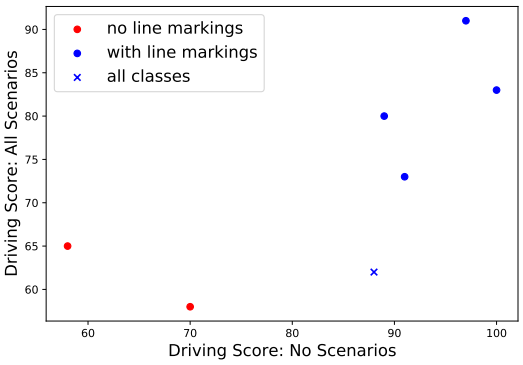


Figure 4.5: Driving scores of all evaluation runs with and without scenarios for different agents trained and evaluated on ground truth data with line markings (blue) and without line markings (red). The model trained on all classes is marked separately.

Although the original paper concludes that agents using semantic label maps as intermediate representation have lower variance than the same models trained with RGB images as input, the driving scores in this experiment still have a large variance. For the results in Table 4.1, the 'Basis' model was trained and evaluated three times with different seeds and the overall driving score varies 9 points. Therefore small changes in the driving score for different agents do not necessarily imply that one class mapping leads to better performance than the other.

Fig 4.5 shows the driving scores of agents, trained with different class mappings for the semantic label maps. The two worst results come from agents trained on class mappings without line markings. All the agents trained with line markings perform much better. This is in line with the results from [24] as they also observed a significant drop in performance as the line markings were excluded from the set of input classes. Additionally, agents trained without line markings have greater difficulties staying in a lane on multi-lane roads. This does not necessarily lower the driving score, as it is not illegal to switch lanes all the time and drive between two lanes. Nevertheless, in practice, this is no desired behavior as it makes it hard for other road users to predict the agent's actions and lowers the driving comfort of passengers.

The worst agent with a separate line markings class is the one trained on all 28 semantic classes. This shows that excluding unnecessary classes does not decrease the overall performance. A similar pattern was observed by Behl et al. [24]. The plot suggests that removing classes from the semantic label maps could lead to higher driving scores. Although, due to the high variance between runs, this is not entirely clear from this experiment. Furthermore, this agent has the highest discrepancy in the scores between the no scenario and all scenarios runs. This again could indicate a higher variance due to irrelevant information in the input images.

The 'No Stop' agent achieves the highest driving score. Since the expert is not able to handle stop points correctly, the agents cannot learn the correct behavior by imitation learning. Therefore separate classes for stop signs and stop line markings probably make it harder for the agent to identify the features important for the driving task.

Overall it is clear that some classes are more important than others when it comes to driving in CARLA. The classes hypothesized by Behl et al. [24] seem to be important in this new setup as well, despite its differences in simulator and model architecture. Still, it is not clear whether the excluded classes are all irrelevant and whether the subset of relevant classes could be reduced further. It is also not clear if there are any classes that should be combined into one class. The following section 4.3 addresses these problems by automatically learning the subset of relevant classes.

4.3 Learning Relevant Classes

From the first experiment, it is clear that some classes are more important than others if semantic label maps are used as input to the driving network. However, it is not clear whether the hypothesized selection of classes is good in the first place. While it is possible to intuitively guess a few relevant classes, there might also be unexpected important classes and correlations between classes that are relevant for the task but hard to guess.

This section demonstrates a method to extract the relevant classes automatically using a genetic algorithm. The essence of this experiment is to show a proof of concept, to get a deeper understanding of what classes are relevant and maybe to make a few unexpected findings. It is not meant to find the one perfect subset of classes as this probably does not even exist.

4.3.1 Setting

Random Color Model

For this experiment, the random color model's encoder architecture is based on a ResNet-18 and it is trained with the same set of hyperparameters as the waypoint model in section 4.2.

To see whether the random color model (RCM) can be used to estimate the performance of an agent trained using a specific class mapping, the RCM is tested using known class mappings from the first experiment.

With the RCM approach, each class mapping evaluation approximation takes around 12 minutes instead of the two days needed for training and evaluating a waypoint model. This is a 240× speedup. The RCM losses in Figure 4.6 show that the 'Basis' and 'No Stop' class mappings achieve a lower loss and therefore score better than the two mappings that do not contain line markings. Further, their scores are very

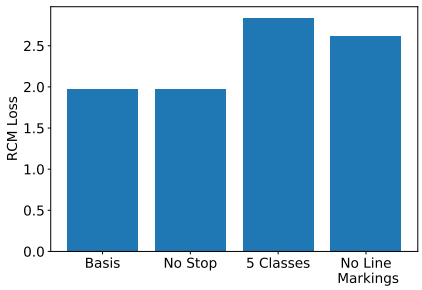


Figure 4.6: RCM losses achieved by different agents on the validation set.

similar which again matches the results from section 4.2. The 'No Line Markings' agent achieves a lower loss compared to the '5 Classes' agent, although in practice their performance was comparably poor. As the simplifications of the RCM lead to more variance on top of the driving score variance, this gap is not surprising. Nevertheless, overall these results suggest that the RCM can be used as a rough approximation for the driving score achieved by different class mappings.

Genetic Algorithm

With the RCM in place, it is still desirable to find new potential class mappings systematically and not only by random guessing. In the following experiment, a genetic algorithm is used to tackle this optimization problem.

The initial population size is set to 800 to start with a wide variety of different mappings. The chromosomes in the initial population are initialized randomly in such a way that they are uniformly distributed over the number of classes to which they map (See Figure 4.8 (a)). Since the RCM is trained on a maximum number of eleven classes, the genes in the chromosomes may range from zero to ten. The reproductive proportion is set to 0.5, so at the end of each generation, the worst 50% of the chromosomes are discarded and do not have a chance to reproduce. The population size for all later generations is limited to 600 to further decrease the run time of the algorithm. The elitism size is 0.05, so for every new generation, the fittest 10% of all chromosomes are directly copied to the next generation. Potential duplicate chromosomes are replaced by random copies of chromosomes from the earlier generation.

After each crossover a random mutation takes place. With a probability of 10%, a gene is flipped either to the class below or above the current value.

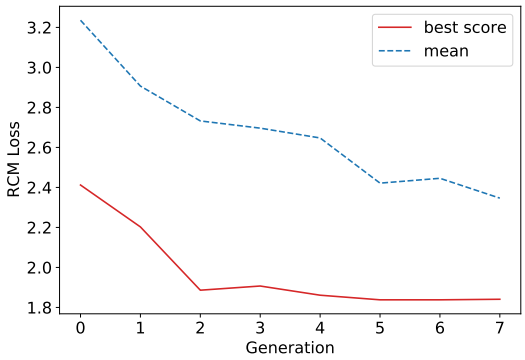


Figure 4.7: RCM losses over time achieved by chromosomes, produced by the genetic algorithm, initialized with a random population. For each generation the lowest loss (best score) (red) and the mean loss over the population (blue) are plotted.

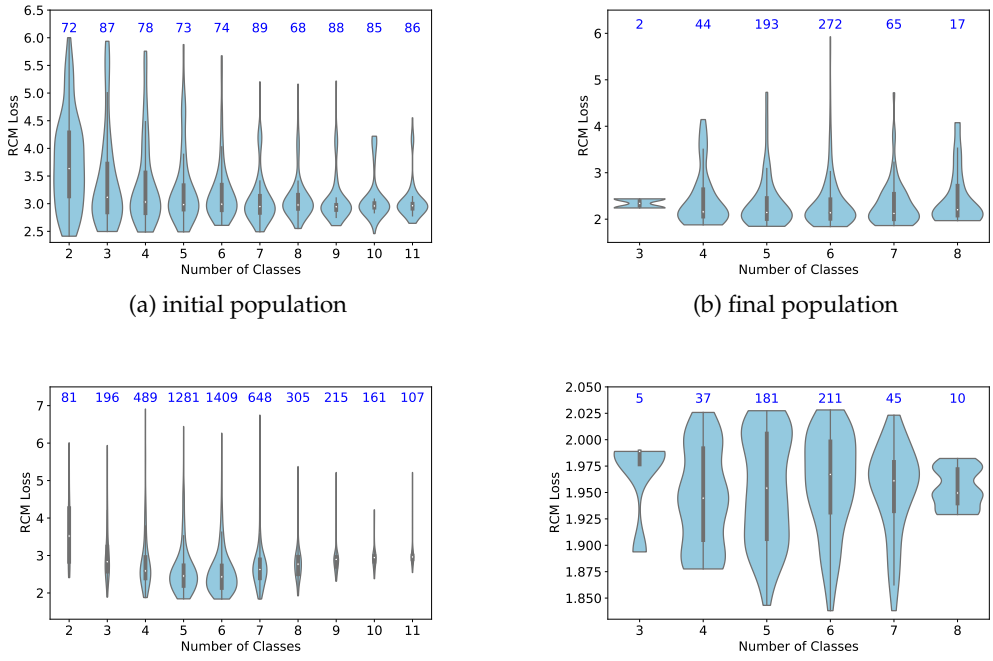
4.3.2 Results

Observations

The implemented genetic algorithm can improve the fitness score of the chromosomes and therefore is capable of finding class mappings that achieve lower RCM losses. Figure 4.7 shows the improvements of fitness scores over time, achieved by the genetic algorithm after random initialization. After only two epochs, the best score decreases from 2.41 down to 1.84. After the second generation, the best score only improves marginally, so it seems to be close to a local minimum. The population's mean score decreases monotonically over all 8 generations except for the plateau between generation 5 and 6 and the overall population mean drops from 3.24 to 2.34, showing that the algorithm is converging.

By design, the initial population is uniformly distributed over the number of classes as seen in Figure 4.8 (a). In the beginning, the top scores are similar for all numbers of classes. The variance of the performances of chromosomes that map to two classes is the highest and with more classes, the variance decreases. The fewer classes there are, the more likely it is that two classes that should be separate are mapped to the same class. This could explain why the distributions are narrower when the class mapping maps to more output classes. In the final population (Figure 4.8 (b)), there are no chromosomes that map to more than eight output classes and no chromosomes that map to only two classes. The algorithm seems to converge to a solution that maps to five to six classes. Overall the shape of the distributions in the final population is more bottom-heavy as in the initial population.

As the goal is to get a deeper insight into which classes matter and what correlations between classes exist and as this optimization problem probably does not have an optimal solution, the algorithm does not have to converge fully towards one



(a) initial population

(b) final population

(c) all chromosomes

(d) top 10% chromosomes

Figure 4.8: RCM loss distributions of initial population (a), final population (b), all chromosomes (c) and the best 10% of all chromosomes (d), sorted by the number of classes to which the respective chromosome maps. The (blue) numbers on top represent the number of chromosomes in the respective bins.

solution. The following results are analyzed statistically over all generations. If the evolutionary algorithm were run for many more generations until it would converge to one solution, this statistical analysis would not work anymore.

When comparing the scores of all chromosomes, generated in all generations, to the scores of the initial population, it is clear that the number of classes is no longer distributed uniformly. Figure 4.8 (c) shows that a large portion of chromosomes map to five or six classes. Running the crossover technique for a few generations without any fitness function creates mostly chromosomes with eleven classes. Therefore this new distribution is not generated by the crossover function itself.

More details can be seen looking at the best 10% of the chromosomes in Figure 4.8 (d). All of the top chromosomes map to at least three and at most eight classes. Considering the variance of this approach, the gap between the best performing chromosomes for five, six, and seven classes is neglectable. However, only 45 of the top chromosomes map to seven classes. All the top chromosomes, shown in the plot, achieve a lower loss than the best chromosome in the initial population.

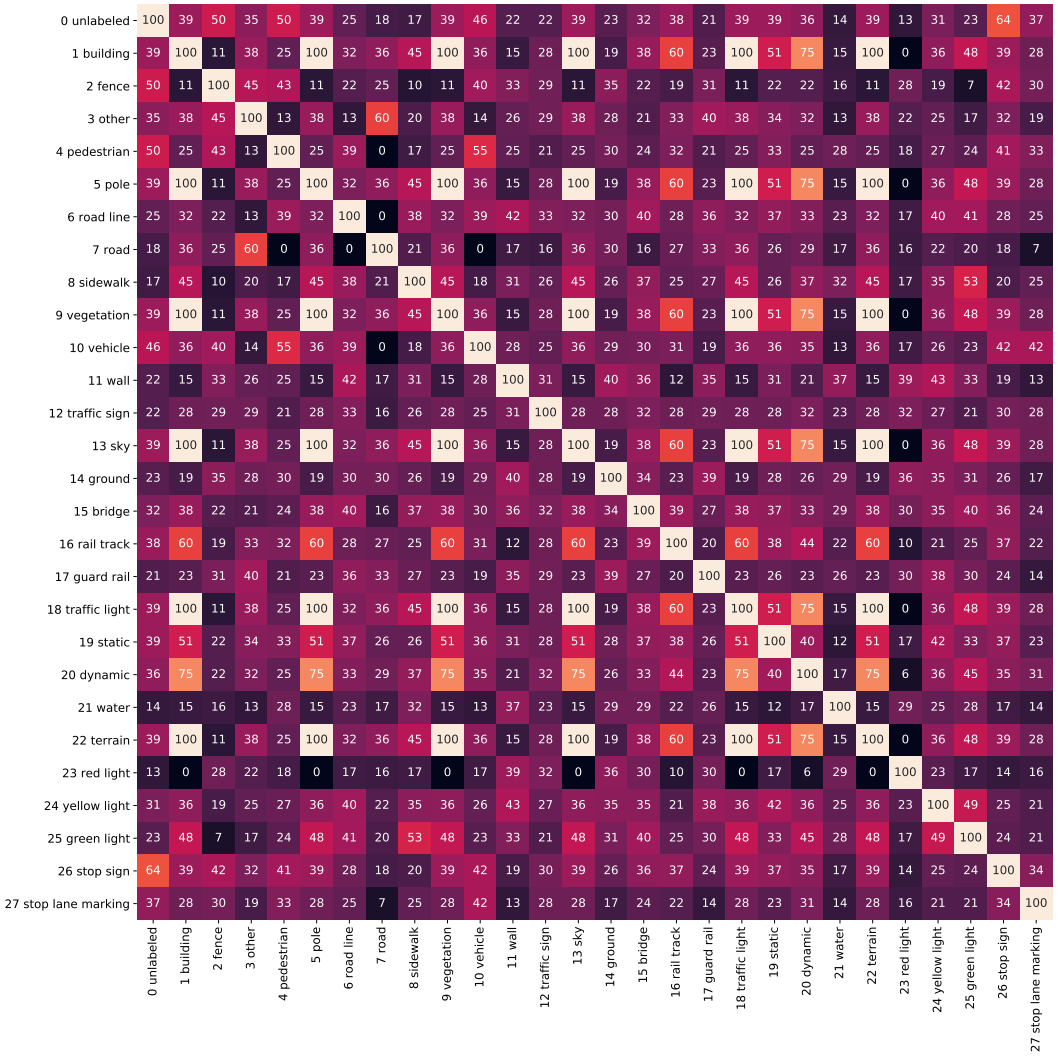


Figure 4.9: The confusion matrix shows the percentages to which an input class is mapped to the same output class as any other input class of the best 10% of all chromosomes.

The confusion matrix in Figure 4.9 shows how often each class is mapped to the same output class as every other class by the top 10% of chromosomes. All the values are presented as percentages. High values suggest that the corresponding classes are mapped to the same class by most of the top-performing chromosomes. Low values show classes that do not get mapped to the same classes very often (if at all). As the matrix displays the top mappings of only one run through the genetic algorithm, not every low or high value necessarily indicates that those classes have some correlation, because the evolutionary algorithm might converge towards a specific local minimum.

Nevertheless, some observations match the experiments from section 4.2. The road class has three zero entries: for pedestrian, road line, and vehicle. This matches the hypotheses from above, as the road should not be in the same class as the aforementioned. Furthermore, looking at all large off-diagonal values, the GA converted towards one output class containing 'building', 'pole', 'vegetation', 'sky', 'traffic light', and 'terrain'. All those input classes are mostly seen in the background of the environment. Mapping those classes into one output class probably removes many details from the top of the images and therefore such a class mapping might reduce distractions for the driving agent.

The 'red traffic light' class is never mapped to the aforementioned large class. If red traffic lights were mapped to this class, many times the agent would not be able to detect them as they might be in front of the sky or a building often. However, one can also see that 'red light' and 'green light' are mapped to the same class in 17% of the cases, which clearly cannot be optimal.

There might not be enough situations in the validation dataset, in which the agent has to detect the traffic light state to make the right decision. Whether an agent has the ability to differentiate red and green traffic lights from each other does not make any difference when the agent is waiting behind another car for a traffic light to turn green. Therefore the images in which the red traffic light truly matters might be underrepresented in the validation set.

The 'fence' class is mapped to the large class by only 11% of the top chromosomes. This value drops down to zero as well, when only the top 5% of chromosomes are considered. When only considering the top 5%, 'dynamic' and 'rail track' are also mapped to the large class more often. Note that there are other classes like 'traffic sign' where this change does not make a huge difference and that the observations above are still visible when considering only the top 5% of chromosomes. This shows that changing this hyperparameter can make a difference in the result for a few classes. The lower this percentage, the bigger the influence of a few local minima, the algorithm converged to.

Figure 4.10 shows a similar confusion matrix plotted for the overall 10% worst chromosomes. In these class mappings, 'vehicle' and 'road' are mapped to the same class in 91% of cases, so having vehicles in the road class might lead to poor results. The large class of the top 10% samples is also slightly visible in this confusion matrix, so having this one large class is probably no guarantee for good performance. The visibility of this class might simply come from the local optimum, the GA converged towards. Maybe 'pedestrian' and 'road line' stand out slightly for the same reason. However, it is noticeable that those two classes are mapped to the road in over a third of the worst class mappings although this never happened in one of the best class mappings.

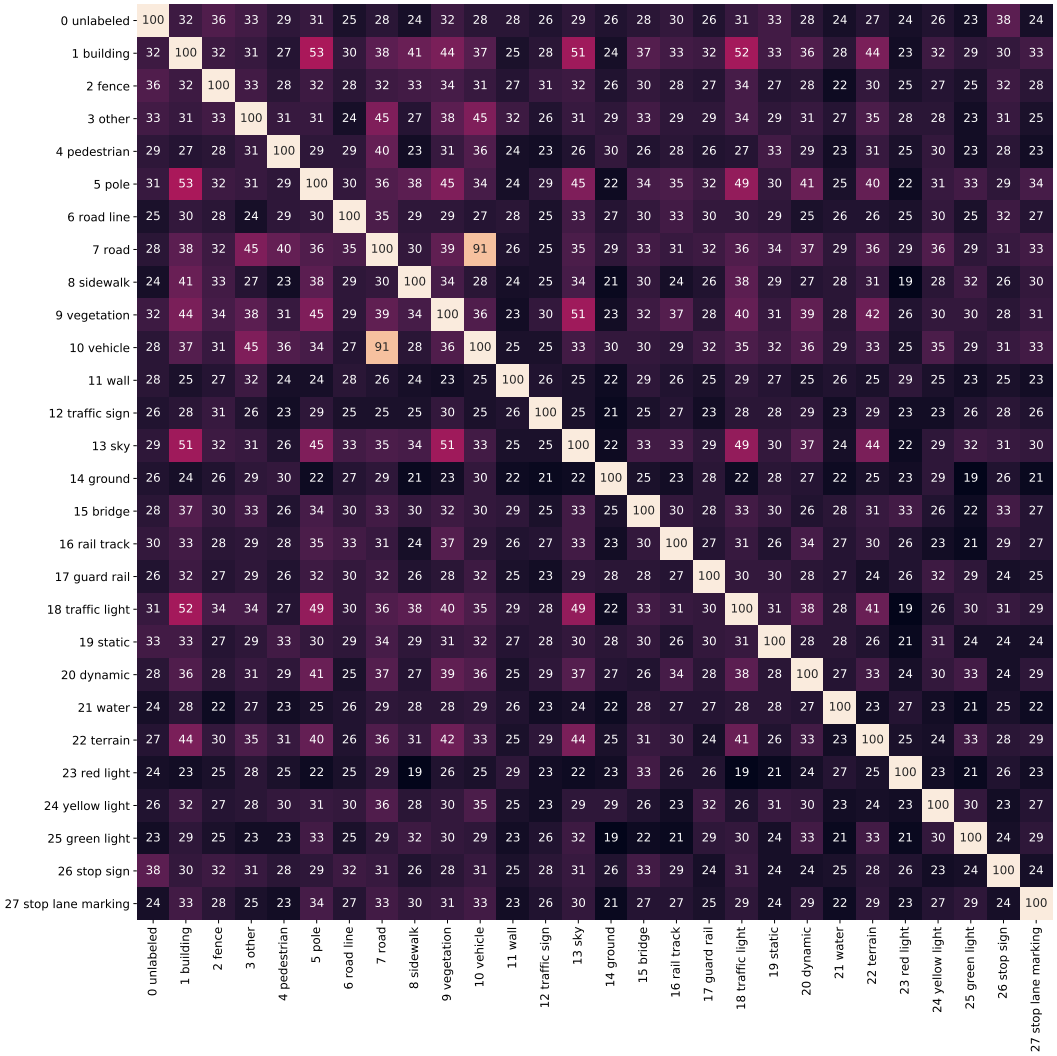


Figure 4.10: The confusion matrix shows the percentages to which an input class is mapped to the same output class as any other input class of the worst 10% of all chromosomes.

Overall it is clear that the GA is able to find good class mappings. Nevertheless, it is unclear which of the mappings lead to better scores and which mappings originated randomly during evolution. Such random mappings do not bring any benefits but might remain in the chromosomes frequently as they do not bring any harm either.

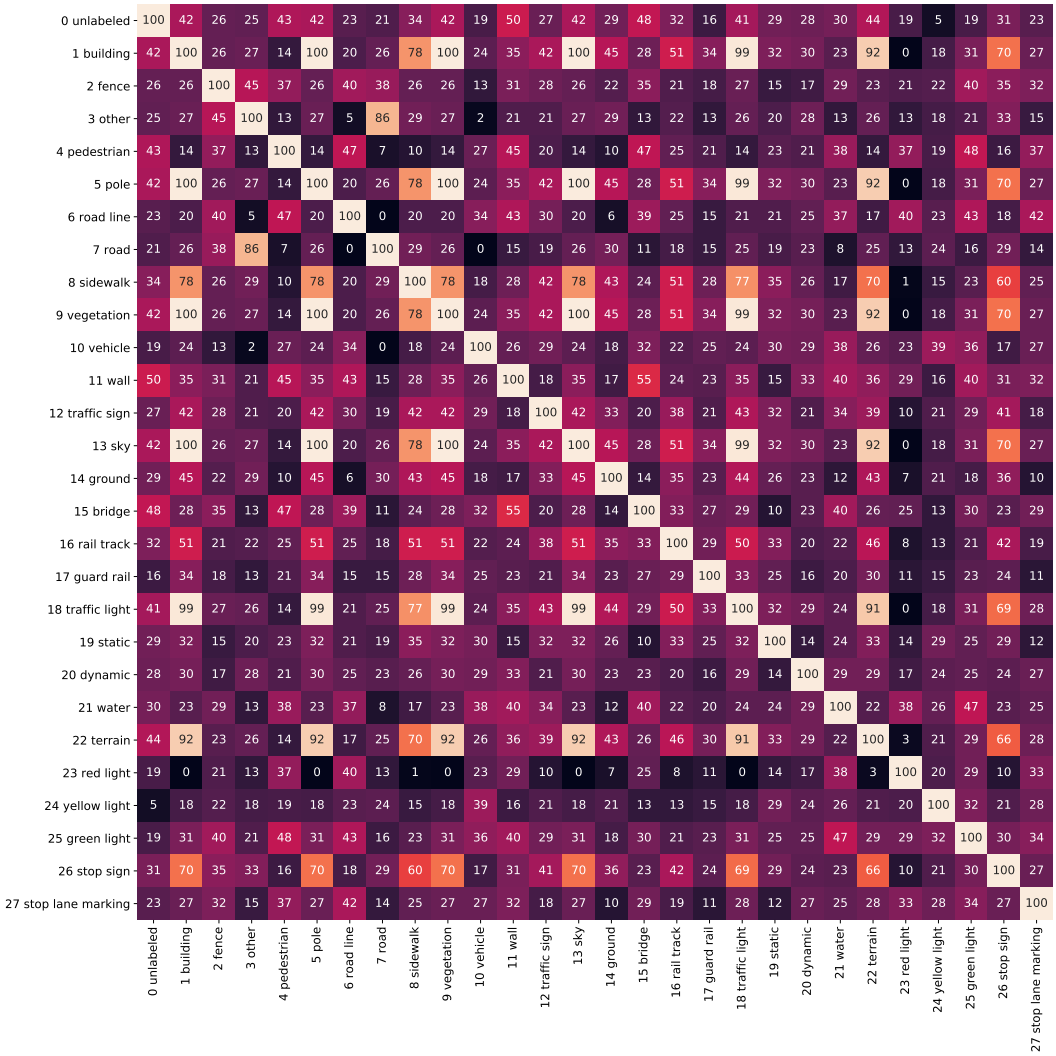


Figure 4.11: The confusion matrix shows the top results of the second run. The elements contain the percentages to which an input class is mapped to the same output class as any other input class of the best 10% of all chromosomes.

Second Run

For further insight, it must be possible to distinguish between actual results and possible random mutations. To get more information about the relevant classes, the GA is run a second time to compare the similarities between the results of the first and second run.

Figure 4.11 shows the top 10% of class mappings, generated by the second run of the genetic algorithm. There are some similarities and some differences compared to

the first run. As in the first run, there is one large class, containing the input classes of 'building', 'pole', and others. This time, 'sidewalk' is frequently mapped to the large class as well. And again, the red traffic light is never mapped to this large class in the top 10% of chromosomes. The surprisingly high value between red and green traffic lights can be observed in this result as well. This time the 'fence' class seems to be equally frequently mapped to the same class as all the other classes except for the class 'road'. The top class mappings rarely map 'road line' to the same class as 'other' and 'ground'.

Considering the mean of the two confusion matrices by taking the mean value for every entry, the similarities can be seen again, whereas the differences tend to cancel each other out. Figure 4.12 shows the (5%) largest and smallest values of this matrix. This way, the similarities between the two runs are visible. The threshold is set to 5%, as this restricts the class mappings in a way that enables the creation of different mappings that map to a maximum of seven classes. Changing this value might again lead to slightly different results.

Interestingly, this makes clear that classes 'road' and 'other' are mapped to the same class by many top-performing chromosomes, so there might be a relationship between those two. The same connection can be seen in the fact that 'road line' and 'vehicle' are rarely mapped to the same class as 'other' as well. Manhole covers are of class 'other' in the CARLA simulator. In some of the towns, there are many such covers all over the roads, so this might be the reason for this connection.

Furthermore, this representation suggests that 'line markings' and 'stop line markings' should not be mapped to the same class as 'road'. Additionally, the 'red traffic light' should not be mapped to the identical class as 'sidewalk', 'rail track', or 'dynamic'. This probably is a remnant of the chromosomes that map those to the large class as well.

The classes not mentioned above are probably not as relevant for the driving task in CARLA. However, there might be classes that do not appear in the figure but do have a small impact on the driving performance. Vice versa, a class is not necessarily really essential for good performance although it is mentioned above, as changing hyperparameters can lead to slightly different results. It might be useful to run the genetic algorithm a few more times and compare the similarities between all runs to extract the essential information. But still, in this experiment, the RCM in combination with the genetic algorithm has shown to be capable of producing intuitively comprehensible results.

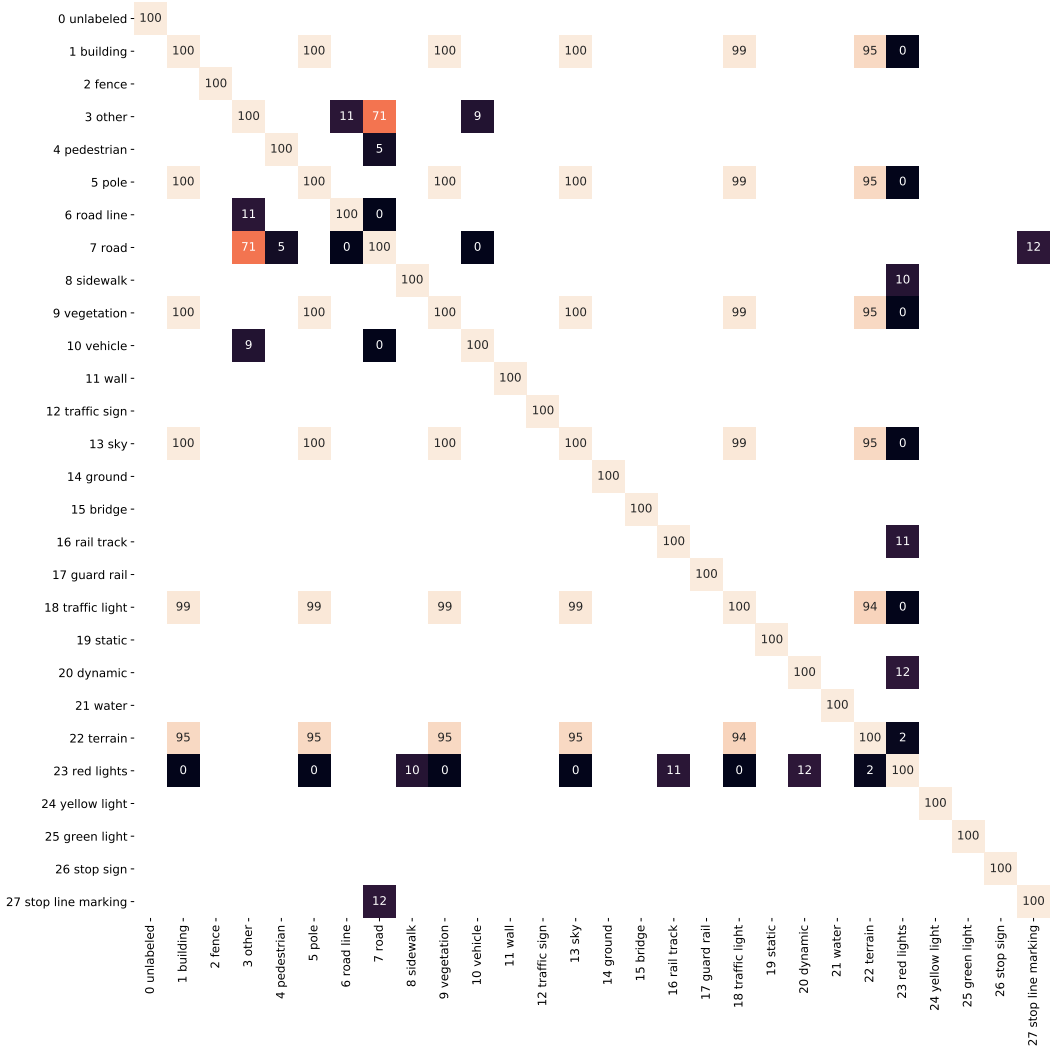


Figure 4.12: Confusion matrix of all class mappings that are similar in the first and the second run of the genetic algorithm. The matrix shows the average percentages, to which an input class is mapped to the same output class as any other input class in the top 10% of the chromosomes from both runs. For better clarity, only the 5% largest and smallest values are shown.

Insights Regarding Class Mappings

Derived from these insights, the following rules for new class mappings are proposed:

1. The class mappings should reduce the number of classes to a maximum of seven.
2. 'road' and 'other should be in the same class.
3. 'pedestrian', 'road line', 'vehicle' and 'stop line marking' should not be in the class 'road'.
4. 'building', 'pole', 'vegetation', 'sky', 'traffic light' and 'terrain' should be mapped to one large class.
5. 'red traffic lights' should not be mapped to this large class as well as to the same class as 'sidewalk', 'rail track', and 'dynamic'.

Based on these rules, a new class mapping can be constructed: as the number of classes should be reduced to seven at most, the first step is to define a new 'unlabeled' class. All the classes the 'red traffic light' should not be mapped to are put in this new class. Further, 'line markings' and 'stop line markings' are mapped to the same class. This is not only done because both should be in class 'road', but also because they are very similar. In fact, the semantic camera in CARLA does not distinguish between the two either. Mapping all irrelevant classes to an additional class reduces the number of classes down to seven.

In practice, the distinction between the irrelevant and the unlabeled class seems unnecessary and is impractical. Mapping them to the same class leads to a total of six classes. This has the benefit that it reduces the labeling effort, as the biggest class does not have to be labeled at all. To check whether these six classes perform as well as the seven classes, two models are trained and evaluated with both class mappings. Table 4.2 shows the driving score and average route completion these models achieved. The model trained on the reduced set of six classes achieved a slightly higher score than the model trained on seven classes. Therefore this simplification seems reasonable and is unlikely to decrease a model's performance.

	7 Classes	6 Classes
Driving Score	54.259	57.932
Route Completion	75.121	82.26

Table 4.2: Comparison of driving score and route completion of the seven learned classes and the simplified subset of six classes.

Overall, the five classes proposed to be most relevant are: 'road|other', 'pedestrian', '(stop) line markings', 'vehicle' and 'red light'. Except for the two classes 'sidewalk' and 'green traffic light', the classes are identical to those proposed by Behl et al [24].

The experiment shows that it is possible to automatically extract relevant classes for using semantic label maps with the presented method. The genetic algorithm in combination with the RCM allows getting a rough overview of which classes might be important and which might be unimportant. However, changes in the hyperparameters, the RCM, and the dataset can lead to slightly different results. Thus, the presented method provides a good starting point for discussion and further research.

4.4 Privileged Experiments

In parallel to the learning of classes (section 4.3), this experiment aims to investigate the influence of the loss function and the size of the encoding ResNet on the driving performance. Therefore, further models are trained using different classes in the semantic label maps. All experiments in this section use ground truth semantic label maps from CARLA for training and evaluation. Unlike the evaluations in section 4.2, all evaluations in this section are executed on the 42 evaluation routes with adversarial scenarios enabled.

4.4.1 Influence of Loss and Encoder

In this section, the influence of the loss and the size of the encoder network on the performance is investigated. For this purpose, four different waypoint models are trained with four different sets of classes: 'All Classes', 'No Stop', '7 Classes', and '2 Classes'. The '7 Classes' agent has the same classes as the 'No Stop' agent, however in this agent, 'other' is in the 'road' class and yellow traffic lights are not in the 'red traffic light' class. The '2 Classes' agent consists of only two classes: 'pedestrian', 'line markings', 'vehicles' and 'red traffic light' are in the first class, all other objects are in the second class. Two models have an encoder based on a ResNet-18, the encoder for the other two models is based on a ResNet-34. Each model architecture is trained once with L_1 and once again with L_2 loss. So overall 16 models are trained and subsequently evaluated.

Figure 4.13 shows a scatter plot of all the models trained in this experiment. On the left are the eight models trained with L_1 loss, on the right are the eight models trained with L_2 loss. As a baseline, the expert score and the score of the RGB waypoint model are shown as well. All models are evaluated on the same routes with adversarial scenarios enabled.

With one exception, all L_2 models are worse than the waypoint model trained on RGB images. This exception seems to be a lucky training seed. It is trained using the 'No Stop' class mapping and the model achieves the highest driving score of all models. Most models trained with L_1 loss achieve a higher score than the RGB model. The results demonstrate clearly that on average, the models trained with L_2

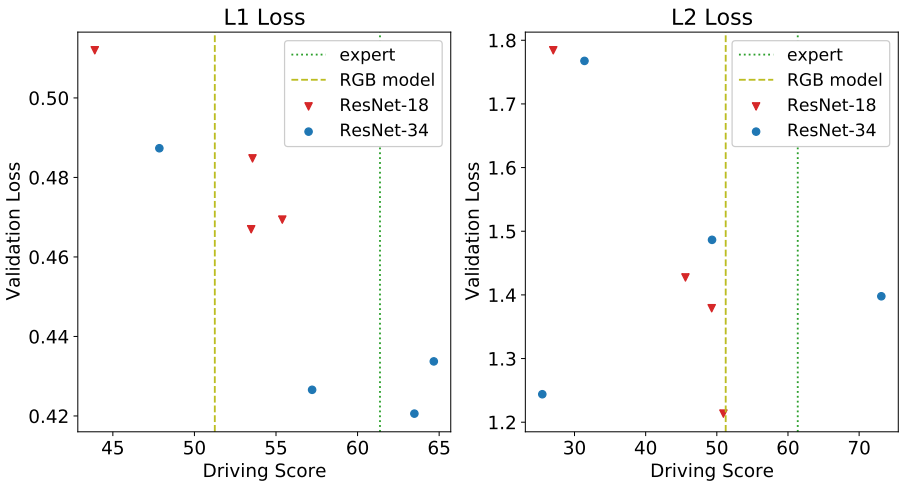


Figure 4.13: Correlation of the validation loss the model achieved during training and the driving score, the respective model scored by evaluation on the evaluation routes with all adversarial scenarios enabled. For comparison, the driving scores of the expert and the RGB waypoint model are displayed as well.

loss achieve a worse driving score than the models trained with L_1 loss. Additionally, there is much more variance in the driving score when the model is trained with L_2 loss. The validation loss and the driving score are more correlated for the L_1 loss than they are when L_2 loss is used. The more the loss and driving score are correlated the better since the loss is the performance defining metric during training.

Considering the two models trained with L_1 loss, the more powerful network on average outperforms the smaller one. Only the '2 Classes' ResNet-34 model is worse than the other ResNet-18 models. The worst performing model trained with L_1 loss is based on ResNet-18.

Overall, these results suggest that even though the waypoint model is trained on semantic label maps, which already greatly reduces the amount of information in the input compared to RGB images, its encoder should be based on a ResNet-34. Furthermore, L_1 loss works better than L_2 loss for waypoint models. This is likely due to the difference in magnitude between the errors of the predicted waypoints and ground truth waypoints. The quadratic error in the L_2 loss possibly leads to an increased influence of the more distant waypoints on the total loss. As a result, the closer waypoints are not learned as well. However, these are essential for a good driving policy.

4.4.2 Excluding Classes

In this experiment, the influence of the remaining classes is examined. To do this, one class at a time is excluded from the 'No Stop' classes. All models are based on a ResNet-34 and are trained with L_1 loss.

The evaluation results in Table 4.3 show that removing the red traffic lights from the classes decreases the performance significantly. This is not surprising as, without this class, there are many situations where there is no way for the model to tell whether there is a red traffic light or not. Removing the vehicles or pedestrian class does not seem to have an impact on the driving score. As they are removed by mapping the respective class to 'unlabeled', the model seems to still be able to detect that there is an obstacle in its way and react accordingly. Both models outperform the expert on the 42 evaluation routes. Excluding the class containing sidewalks does not seem to negatively impact the performance either.

	No Traffic Light	No Road	No Vehicle	No Pedestrian	No Sidewalk
Driving Score	38.551	61.583	61.186	59.294	62.238

Table 4.3: Driving scores achieved by different models. All models are based on the 'No Stop' classes, but for each model a different class is excluded. The models are evaluated on the 42 evaluation routes.

4.5 Additional Affordances

The goal of this experiment is to investigate whether learned affordances are a suitable form of intermediate representation for relevant information and whether adding affordances to the waypoint model can improve the performance. Therefore affordances for vehicles and red traffic lights are added separately. The expert has an additional pedestrian affordance, but these are ignored in this section because avoiding pedestrians seems to be an easy problem in the evaluation routes used, as there are many evaluations without any pedestrian infraction. Therefore there are not enough pedestrian infractions to be sure that the infractions are not just due to bad luck. For these experiments, the models from earlier experiments are reused, as the affordances are added during deployment and therefore do not change the training process. In this experiment, all affordances are considered true if the respective output value of the model is higher than 0.5, otherwise, they are considered false. If any affordance is true, the agent is stopped immediately.

For this experiment, the ground truth semantic label maps are used as well. However, the affordances are produced by an affordance model from the RGB camera images.

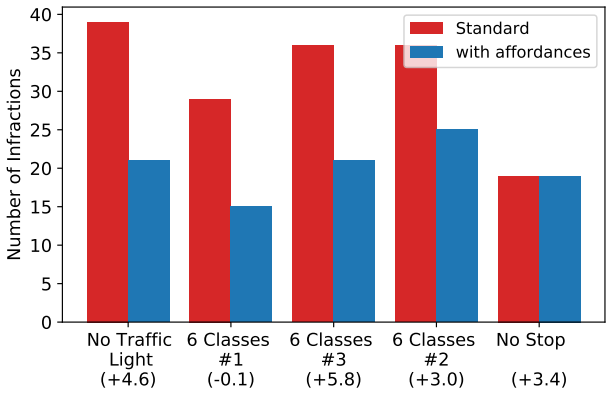


Figure 4.14: Number of traffic light infractions on the evaluation routes. With additional traffic light affordances, the number of traffic light infractions is reduced. The numbers within the brackets represent the change in driving score after adding traffic light affordances.

4.5.1 Traffic Light Affordances

Adding a traffic light affordance to the 'No Traffic Light' agent reduces the number of traffic light infractions from 39 down to 21 and thereby increases the driving score by 4.6 points to 43.128. Therefore, the 'No Traffic Light' agent with additional traffic light affordance is still significantly worse than the baseline RGB waypoint model.

Since the model is not able to recognize traffic lights, it cannot distinguish the situations in the data set when the expert waits at a red light from those when the expert does not stop at a green light. However, since waiting at an intersection takes much longer than driving through an intersection, there are many examples in the dataset in which the agent waits at intersections. Therefore the 'No Traffic Light' agent learns both, to stop and to drive through intersections and the agent's behavior at intersections is unpredictable. Adding traffic light affordances stops the agent from running red lights but does not help the agent to drive on at green traffic lights. Therefore its driving score is still relatively low. Training the 'No Traffic Light' agent on a subset of the dataset, in which all samples with red traffic lights are excluded might solve this issue. Alternatively, it would be possible to provide the model with the affordance as an additional input, so that it can distinguish red from green traffic lights during training.

Adding the traffic light affordance to models whose semantic label maps already contain traffic lights reduces the number of traffic light infractions and improves the agent's performance. Figure 4.14 shows the number of traffic light infractions for models with and without additional traffic light affordance. On average, the additional traffic light affordance increases the driving score by 3.3 points. However, there are cases where the improved performance at traffic lights can lead to a slightly

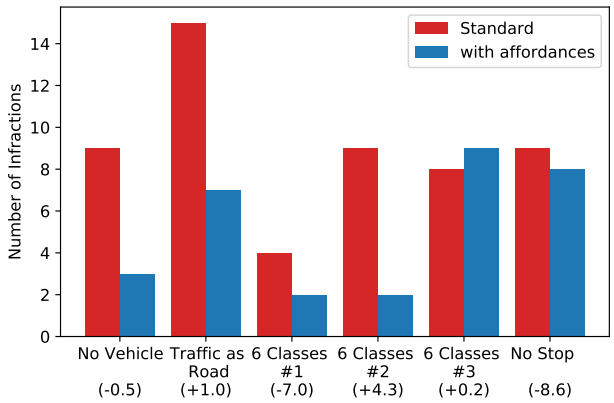


Figure 4.15: Number of collisions with other vehicles on the evaluation routes. With additional vehicle affordances, the number of collisions is reduced in all but one case. The numbers within the brackets represent the change in driving score after adding vehicle affordances.

lower driving score because on some routes the agents might time out due to waiting at red traffic lights. The 'No Stop' agent has 19 traffic light infractions with and without additional affordance. Therefore, its improvement in driving score cannot be attributed to the additional traffic light affordance.

One reason for this improvement might be that the affordance model predicts the affordance based on the full resolution of the input image whereas the waypoint model uses cropped images for its input. However, there are situations where the waypoint models slowly runs over red lights although they are visible. In these situations, the traffic light affordance helps to stop the vehicle until the traffic light is green.

4.5.2 Vehicle Affordances

Adding vehicle affordances does not improve the performance of the models. Figure 4.15 shows the number of vehicle collisions with and without additional vehicle affordance for different models. Although for most agents, there are fewer collisions with other vehicles, on average, the driving score drops by 1.8 points with the additional vehicle affordances.
For the 'Traffic as Road' agent, vehicles are set to the class 'road' instead of unlabeled and therefore they are invisible to the agent. This only makes sense for testing purposes, as the classes differ greatly in reality and all vehicles would have to be actively added to the road class when labeling. However, a 'No Stop' agent altered this way, achieves a driving score of only 31.4. Adding the vehicle affordance to this agent insignificantly improves the driving score to 32.4, although the number of vehicle collisions is reduced by 50%.

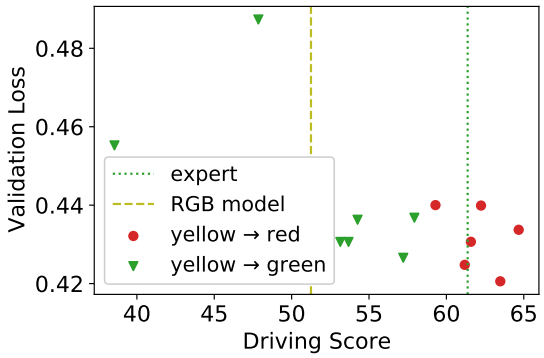


Figure 4.16: Validation loss and driving score of all ResNet-34 models trained with L_1 loss. The best performing models are all trained to stop for yellow traffic lights.

4.6 Further Investigations

Figure 4.16 shows a scatter plot of all evaluations of the ResNet-34 models trained with L_1 loss and their driving score without additional affordances. Except for the two outliers ('2 Classes' and 'No Traffic Light'), all models outperformed the RGB baseline model. It is noticeable that the five best performing models (red circle) are all trained with class mappings that map yellow traffic lights to the same class as red traffic lights, whereas all the other models (green triangle) are trained with semantic label maps in which the yellow traffic lights are classified as unlabeled.

In practice, this seems to be a very important feature, even though it is not visible in the validation loss. This means that the RCM in its current version is not able to detect the correlation between the red and yellow traffic light classes.

Ignoring the two outliers, there only is a marginal correlation between driving score and validation loss for the top-performing models, if any correlation at all. The best performing model even has a higher loss than many of the other models. However, the difference in validation loss between the models might be neglectable considering the variance during training. This is not only a problem for the search of relevant classes using the RCM, but also for the training itself, as the models are trained with this L_1 loss and at the end of the training, the weights that lead to the lowest validation loss are selected as optimal.

Extensively studying the average L_1 loss, unraveled separately for many different specific situations such as red traffic lights and speed limits as well as broken down by expert actions, did not lead to an improved correlation between the driving score and the loss. Looking only at the worst few percent of examples of such situations has not helped either.

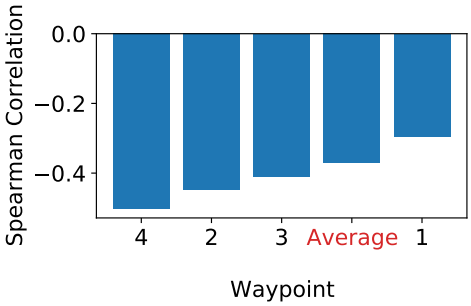


Figure 4.17: Spearman correlation between L_1 loss and driving score, broken down by the separate waypoints. The average over all waypoints (red) is used for the model selection after training. All data points show a negative correlation, therefore a lower loss corresponds to a higher driving score. However, the correlation is weak and probably insignificant for all data points except for the fourth waypoint.

Figure 4.17 shows the Spearman correlation between the driving scores and the models' losses broken down by waypoints. The correlation is highest for the fourth waypoint, which is the farthest from the vehicle and lowest for the first waypoint. During training, the waypoint average is used to calculate the loss. In a future experiment it might be interesting to check whether there is a performance difference between two models from the same training run, but one selected by the lowest validation loss over all waypoints and the other one selected by the lowest validation loss for only the fourth waypoint.

Overall, finding a loss metric that correlates better with the driving score would be an important task for future work. Until there are new findings, the L_1 loss over the entire validation set remains the best approximation to the driving score.

4.7 Non Privileged Experiments

The goal of this experiment is to find out if it makes a difference whether ground truth semantic label maps from CARLA or segmentations from a segmentation model are used for training and evaluation.

To test the influence of privileged information on evaluation, the agents use the ground truth semantic label maps provided by CARLA as input during training. After training, each agent is evaluated twice: once with the privileged semantic representation from CARLA and again with the non privileged semantic representation, generated from RGB camera images by an upstream segmentation model.

Figure 4.18 shows the driving scores the agents achieve in the two evaluation runs. Most models perform slightly worse with non privileged semantic label maps as

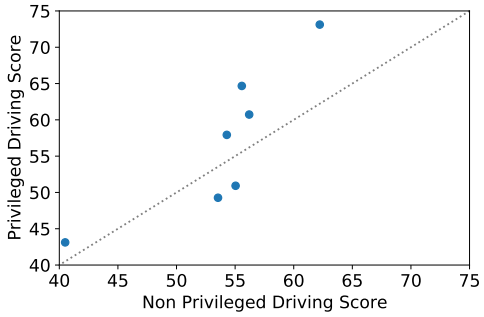


Figure 4.18: Driving scores of models that are evaluated with privileged and non privileged segmentation representations as input. The dotted center line marks all points with identical driving scores.

input. The driving scores show a strong correlation and have a Spearman correlation coefficient of 0.93. Therefore, the conclusions from privileged experiments can be well applied to agents that have no access to privileged semantic label maps. This finding may be useful for further experiments, as the segmentation network requires additional computing power, which slows down the evaluation.

In this experiment, the same '7 Classes' model is trained two times: once with ground truth segmentation inputs from CARLA (Hybrid) and once with non privileged semantic label maps generated by the segmentation model (Standard). Both models are evaluated with the non privileged semantic label maps.

	Hybrid	Standard
No scenarios	72.55	67.28
All scenarios	52.13	52.43
Overall	62.35	59.86

Table 4.4: Driving scores achieved by the same '7 Classes' model, trained with ground truth segmentation inputs from CARLA (Hybrid) and with semantic label maps produced by the segmentation model (Standard). For evaluation, both models use non privileged semantic label maps.

The results shown in Table 4.4 suggest that models can be trained with ground truth semantic label maps, although, when deployed, the intermediate representation is generated by a segmentation model. Overall, the Hybrid approach even slightly outperformed the Standard approach. This matches the observations by Gharaee et al. [39]. This can be useful, as training the models with ground truth segmentation inputs can simplify and speed up the training process. However, further tests should be performed to be sure about this.

4.8 Visual Abstractions

From the previous experiments, it is clear that the waypoint model's performance is improved when it utilizes semantic label maps, containing relevant classes, as intermediate representation instead of directly inferring waypoints from RGB images. In this section, the difference in the performance of the baseline models (described in section 4.1.5) trained with and without semantic label maps as visual abstractions, is investigated.

For this experiment, each of the models is trained and evaluated once with RGB images and once again with semantic label maps containing the '6 Classes' described in section 4.3. Each model is evaluated on the 42 evaluation routes with adversarial scenarios enabled. For this comparison, privileged semantic label maps from CARLA are used.

Architecture	RGB	Visual Abstraction
Waypoint Model*	51.24	54.91
LBC	29.07	43.83
CILRS	25.06	30.73
Full Affordance	46.23	49.41

Table 4.5: Driving scores for different architectures trained with and without visual abstractions. Evaluation is done on the 42 evaluation routes. Ground truth semantic label maps with '6 Classes' are used as visual abstraction. (*) For the waypoint model's driving scores, the mean over three evaluations is used

The results in Table 4.5 show that all models benefit from using semantic label maps provided by CARLA as input. All models achieve a higher driving score when trained on the semantic representation compared to the same models trained with RGB camera images.

Some architectures might benefit more from the ground truth information encoded in the semantic label maps than others. On the other hand, using a better suited set of classes instead of the '6 Classes' used in this experiment might further improve the benefit of using the semantic representation. Future research should therefore investigate the impact of non privileged semantic label maps on the performance of those models.

5 Conclusion

This work demonstrates that semantic segmentation can be used as an effective intermediate representation to improve the performance of imitation learning based autonomous vehicles. When using this approach, only a few specific classes are relevant for the driving task and therefore needed in the semantic representation. Excluding irrelevant classes from this representation not only improves the agent's performance but can also drastically reduce labeling costs.

In order to automatically extract these relevant classes from a set of classes for a given task, the new 'random color model' is proposed. In combination with a genetic algorithm, the model can be used to identify classes that are important for the task. In a proof of concept, this method shows that, without any presupposed knowledge of the task, it can find a set of relevant classes that, when used in the presented waypoint model that makes use of segmentation-based visual abstractions for driving in CARLA, reliably outperforms recent end-to-end methods in challenging multi-town multi-weather settings.

Adding learned affordances to additionally encode important environment attributes reduces the number of infractions. However, they need to be carefully selected to be sure to improve the performance. While an agent benefits in most cases of an additional affordance, encoding the traffic light state, other affordances are shown to even reduce the agent's performance.

The empirical study shows that even if the complexity of the input is reduced by visual abstraction, this does not mean that the size of the model can be reduced. Furthermore, the analysis shows that for training the model used in this work, L_1 loss is preferable to L_2 loss. However, looking at the models that perform well, the L_1 validation loss only correlates weakly with the driving score that the models achieve in the evaluation. Finding a better suited offline evaluation metric that has a higher correlation with the driving score would be an important task for future work.

This work provides insights into how visual abstraction can be used for autonomous driving and what needs to be considered when learning driving policies with state-of-the-art imitation learning methods.

List of Figures

List of Tables

Bibliography

[1] N. Larco. (2018). "How will autonomous vehicles transform our cities?" TED, [Online]. Available: https://www.ted.com/talks/nico_larco_how_will_autonomous_vehicles_transform_our_cities.

[2] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., 2015.

[3] A. Efrati. (2020). "Money pit: Self-driving cars' $16 billion cash burn," [Online]. Available: https://www.theinformation.com/articles/money-pit-self-driving-cars-16-billion-cash-burn (visited on 10/09/2020).

[4] J. Janai, F. Güney, A. Behl, and A. Geiger, *Computer vision for autonomous vehicles: Problems, datasets and state of the art*, 2021. arXiv: 1704.05519 [cs.CV].

[5] T. Bräunl, *Embedded Robotics - Mobile Robot Design and Applications with Embedded Systems*. Springer, 2008, p. 416.

[6] E. Dickmanns and A. Zapp, "Autonomous high speed road vehicle guidance by computer vision1," *IFAC Proceedings Volumes*, vol. 20, no. 5, Part 4, pp. 221–226, 1987, 10th Triennial IFAC Congress on Automatic Control - 1987 Volume IV, Munich, Germany, 27-31 July, ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)55320-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017553203.

[7] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the carnegie-mellon navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 362–373, 1988. DOI: 10.1109/34.3900.

[8] *Sae international j3016; levels of automated driving graphic*, https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic, Accessed: 2021-04-14.

[9] *Waymo is opening its fully driverless service to the general public in phoenix*, https://blog.waymo.com/2020/10/waymo-is-opening-its-fully-driverless.html, Accessed: 2021-04-14.

[10] Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim, "An analysis of adversarial attacks and defenses on autonomous driving models," *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10, 2020.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[12] Epic Games, *Unreal engine*, version 4.24. [Online]. Available: https://www.unrealengine.com.

[13] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, *Solving rubik's cube with a robot hand*, 2019. arXiv: 1910.07113 [cs.LG].

[14] *Rethinking cruise's av development loop during covid-19*, https://medium.com/cruise/cruise-av-development-loop-covid-19-1daef2f0c3d5, Accessed: 2021-04-23.

[15] *Off road, but not offline: How simulation helps advance our waymo driver*, https://blog.waymo.com/2020/04/off-road-but-not-offline--simulation27.html, Accessed: 2021-04-23.

[16] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, *End-to-end driving via conditional imitation learning*, 2018. arXiv: 1710.02410 [cs.RO].

[17] M. Martinez, C. Sitawarin, K. Finch, L. Meincke, A. Yablonski, and A. Kornhauser, *Beyond grand theft auto v for training, testing and enhancing deep learning in self driving cars*, 2017. arXiv: 1712.01397 [cs.CV].

[18] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, *Playing for data: Ground truth from computer games*, 2016. arXiv: 1608.02192 [cs.CV].

[19] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, *Deepdriving: Learning affordance for direct perception in autonomous driving*, 2015. arXiv: 1505.00256 [cs.CV].

[20] *Dmv disengagement reports*, https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/disengagement-reports/, Accessed: 2021-04-09.

[21] 2. C. A. D. Leaderboard. (2020). "2020 carla autonomous driving leaderboard," [Online]. Available: https://leaderboard.carla.org (visited on 11/20/2020).

[22] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed., Morgan-Kaufmann, 1989, pp. 305–313. [Online]. Available: http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf.

[23] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," vol. 1604.07316, 2016.

[24] A. Behl, K. Chitta, A. Prakash, E. Ohn-Bar, and A. Geiger, *Label efficient visual abstractions for autonomous driving*, 2020. arXiv: 2005.10091 [cs.CV].

[25] S. Ross, G. J. Gordon, and J. A. Bagnell, *A reduction of imitation learning and structured prediction to no-regret online learning*, 2011. arXiv: 1011.0686 [cs.LG].

[26] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[27] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," 2019.

[28] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," 2018.

[29] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, 2015. arXiv: 1411.4038 [cs.CV].

[30] F. Sultana, A. Sufian, and P. Dutta, "Evolution of image segmentation using deep convolutional neural network: A survey," *Knowledge-Based Systems*, vol. 201-202, p. 106 062, 2020, ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2020.106062. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705120303464.

[31] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[32] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[33] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang, "The apolloscape dataset for autonomous driving," *arXiv: 1803.06184*, 2018.

[34] M. Contributors, *MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark*, https://github.com/open-mmlab/mmsegmentation, 2020.

[35] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Conference on Robot Learning (CoRL)*, 2019.

[36] A. Zhao, T. He, Y. Liang, H. Huang, G. V. den Broeck, and S. Soatto, *Sam: Squeeze-and-mimic networks for conditional visual driving policy learning*, 2020. arXiv: 1912.02973 [cs.CV].

[37] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 4, pp. 640–651, 2017.

[38] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[39] Z. Gharaee, K. Holmquist, L. He, and M. Felsberg, *A bayesian approach to reinforcement learning of vision-based vehicular control*, 2021. arXiv: 2104.03807 [cs.CV].

[40] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014. arXiv: 1406.1078 [cs.CL].

[41] N. J. A. Sloane and T. O. F. Inc., *The on-line encyclopedia of integer sequences*, 2021. [Online]. Available: https://oeis.org/A000110.

[42] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975, second edition, 1992.

[43] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020, ISSN: 2168-2275. DOI: 10.1109/tcyb.2020.2983860. [Online]. Available: http://dx.doi.org/10.1109/TCYB.2020.2983860.

[44] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–

8126, Oct. 2020. DOI: 10.1007/s11042-020-10139-6. [Online]. Available: https://doi.org/10.1007/s11042-020-10139-6.

[45] K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," Tech. Rep., 1975.

[46] A. Sauer, N. Savinov, and A. Geiger, *Conditional affordance learning for driving in urban environments*, 2018. arXiv: 1806.06498 [cs.RO].

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.