UiA University of Agder

Road Detection as a part of the Reward System for Reinforcement Learning-based Autonomous Cars

MARTIN HOLEN

SUPERVISOR
Morten Goodwin

University of Agder, 2020
Faculty of Engineering and science
Department of ICT

Master

Abstract

Human drivers are subject to numerous flaws. It is common that the drivers get tired and lose control of the vehicle, and some even get drunk or high, which results in dangerous situations for themselves and others.

Autonomous driving is a field of study which has gained notoriety lately, as it attempts to get more reliable than humans at driving; Though there is still much research to be done in the field. We are far away from replacing human drivers with safe AI-equivalents.

In this thesis, we aim to validate a road detection algorithm as a part of the reward system of the autonomous vehicle[1]. We introduce a road detection as a simple supervised model, which predicts if the car is off or on the road, and is specifically designed to support Reinforcement Learning Algorithms such as Proximal Policy Optimization, Deep Q-Networks, and Deep Deterministic Policy Gradient.

[1] 'Road Detection for Reinforcement Learning-based autonomous cars' is included as a part of this thesis, it is also accepted in ICISS 2020

Acknowledgements

First I would like to thank my main supervisor, Associate Professor Morten Goodwin Ph.D. at the ICT Department at UiA for all of his guidance and help. I'd also like to thank my co-authors from the Road Detection for Reinforcement learning based autonomous cars paper, for all of their invaluable help. As well as this, I wish to thank UiA for allowing me to use their Nvidia DGX-2 servers to run my code, and UiA for all their resources.

Abbreviations

AI - Artificial Intelligence

NN - Neural Network

CNN - Convolutional Neural Network

DNN - Deep Neural Network

RL - Reinforcement learning

RNN - Recurrent Neural Network

LSTM - Long Short Term Memory

PPO - Proximal Policy Optimization

DDPG - Deep Deterministic Policy Gradient

DQN - Deep Q Networks

ResNet - Residual Neural Network

MNASNet - Mobile Neural Architechture Search Network

FPNAS - Fast and Practical Neural Architechture Search

TRPO - Trust Region Policy Optimization

IMU - Inertial Measurement Unit

GPS - Global Positioning System

List of Figures

List of Tables

Contents

1 Introduction

Driving is a dangerous thing to do, as 1.3 million people die every year in traffic [1], most of which are caused by human error. Some tragic event causes are drunk driving, people getting tired, and road distractions. Autonomous cars do not suffer from the same issues, which has led researchers to attempt to create a fully autonomous car. Companies have joined in the pursuit of autonomous vehicles, of which Tesla is a significant brand whose data shows that cars using their autonomous features to be safer than human drivers [2]. The topic of self-driving or autonomous vehicles is a complex one, as the vehicles need to follow the traffic laws, on top of this, the vehicles need to be able to predict humans and animals who may not always be following the traffic laws. As well as humans and animals on the road, there is also the issue of traffic laws and traffic markings which differ between countries.

In this thesis we introduce a road detection system as a way to aid autonomous cars trained using Reinforcement Learning. We test this in two environments, namely a Unity environment and a Gym environment. The proposed road detection system uses a Supervised Learning model to give a penalty when the car drives off the road. The road detection model is trained in one environment and then using Transfer Learning can quickly be retrained to be used in another environment.

There are many ways of creating self-driving cars, one common method is called Supervised Learning. Supervised learning works by giving the AI an input, such as an image from a camera, and given this input the AI tries to predict the output; The output could be a segmented image and some actions. The AI receives the output and tries optimizing its algorithm to correct its prediction by moving closer to the correct output. Among the supervised learning methods, the most relevant subcategories for autonomous vehicles are classification and segmentation. As classification classifies the steering angle, and how far the accelerator is pressed. And the segmentation could be

used to change the input image, by separating the different objects in the image to different colors.

Supervised learning is a big field, consisting of many methods, such as statistical machine learning and Neural networks. The statistical implementations include methods such as Bayesian Classifier and Decision Trees. These methods function by taking inputs and predicting classes, and their predictions are based on the likelihood given a set of features also called the input. These are created for a specific task and have some issues when it comes to generality. Then there are the neural network-based implementations. These methods can take many sensor inputs and create a generalized set of actions to perform, given these sensor inputs. Though generality for these methods is good, though, there are still issues; that will be discussed later.

A new field of research for self-driving cars is that of Reinforcement Learning. This field consists of finding the best action to take for any environment, often using deep neural networks. In order to find the best action to take, RL algorithms need to try different actions, leading to the issue of exploration versus exploitation [3]. Deep RL has the generality required for autonomous cars though there are cons to it, these will be explored later.

1.1 Background

Self-driving cars have had a large variety of methods, from mathematical implementations such as vanishing point calculations [4], to more complex ones such as implementing a lane keeping algorithm which learns to drive in a day [5].

1.1.1 Neural Networks

Amongst AI, one of the most prominent methods in recent years is Neural Networks. The simplest Neural Networks are based on a simplification of how neurons in the brain function, by taking inputs and multiplying them by weights before adding a bias, this is called a neuron. Adding multiple layers of neurons together allows these algorithms to solve intricate patterns.

At any point in the Neural Network, one can also add activations. Activations change the values of the outputs to aid in pattern recognition. Two commonly used methods are called ReLU or Recurrent Linear Unit, and Softmax. ReLU is a simple activation. ReLU checks if the value is below 0, if the value is below 0, ReLU sets the value to 0 instead; otherwise, it stays the same. This activation removes the issue of getting large negative numbers, which ruin the prediction, as for classification, each output has a probability of being the correct output, and therefore has a value above 0. For Softmax, the activation makes each output between 0 and 1, with the sum of all of the outputs being 1. When the sum of the outputs is 1, it means that the output is the percentage probability that the specific output is correct.

There are also more complex Neural Networks, some of which are more easily able to find patterns in larger inputs, such as high-resolution photos, as well as ones able to take into account what has happened previously. These Neural Networks are called Convolutional Neural Networks (CNN) and Recurrent Neural Networks(RNN), respectively. The CNNs are based on the visual cortex of animals function, looking at patterns of different sizes [6]. This functions by creating a "Kernel" of a set size, typical examples include 3x3, and for this kernel, create a weight for each instance. A kernel of size 3x3 has 9 weights, each of these weights is multiplied by an input, and summing the outputs for one kernel now becomes that output. There is also what is called a filter size, which determines the channel size. This channel size can be thought of like the colors of the image, so by sending in an RGB image and setting the filter size to 32, the photo goes from 3 channels to 32 channels. One can also use a bigger stride, meaning that the

kernel is moved more than one pixel at a time. This allows for the CNN to search for a pattern which may have some overlap, but not necessarily a lot. For Recurrent Neural Networks, the main focus is to find patterns that occur over time. The patterns that occur over time could be to predict where an object is moving. As RNNs take into account what happened in the last few time-steps, the RNN may be able to find out what is going to happen from small patterns that occur over time, such as a person leaning in for a kiss or speech recognition. It can do this in multiple different ways, and a quite common one is called LSTM or Long Short Term Memory, which uses a forget gate, which could mean that happened a long time ago is less relevant that something which happened in the previous time interval [7].

1.1.2 Learning

When considering Neural Networks (NN), one of the commonly used methods for training the NN is supervised learning. Supervised learning revolves around getting inputs and predicting an output; this predicted output is the check against a ground truth. The difference between the prediction and the ground truth is called the loss. This loss refers to how well the algorithm did at predicting the correct output. So if the algorithm was supposed to predict three, and it got one, then the loss is two as it missed the correct output by two. The algorithm tries improving itself so that its output now becomes three, the action of doing this is called backpropagation. This backpropagation algorithm checks how right the prediction was at every level of the algorithm. If one layer was mostly to blame for the poor prediction, then that layers weights will be changed most, so that the next time the network predicts it predicts closer to the correct output. If the loss is used without alterations, however, there is an issue. As one input may change the network to predict one thing, and the next input changes the network entirely again. To solve the issue of the network changing completely for each input it gets, a learning rate is used. This learning rate takes the loss and multiplies it by a number smaller than one. The result of this is that when the Neural Network is training, it does not undo all of its training just because it gets a different set of inputs;

instead, the NN tries to move towards the correct action. There is also another way of training that is commonly used, especially when it comes to beating video games, called Reinforcement Learning (RL). Reinforcement learning (RL) is a bit different in how it works when compared to supervised learning, as it performs an action and gets a reward for the action. The RL algorithm then tries to maximize the reward that it gets, and this can be done in many different ways. One of these methods is called a Deep Q Network (DQN), the DQN, predicts the reward it will get at any time, and then checks how good its predictions are. If the predictions are not correct, then the network changes its weights to improve the prediction of the reward. One method of doing so is by using the Bellman Equation, seen in table 1. The Bellman equation calculates the expected reward and is used in some notable RL algorithms, such as DQN [8]. There are also some policy gradient algorithm such as PPO and DDPG, which attempts to create the optimal policy. These policies are updated frequently to check how good they are, and improving them based on their performance [9][10].

$$Q*(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)] \tag{1}$$

Table 1: Bellman Equation

The issues with supervised DNNs is that of data. Supervised NNs need large amounts of perfect data, if they are to perform the correct actions. This data also needs to be general, meaning that it includes different environments so that the NN does not simply train for one environment. Training in a desert environment might make the algorithm work well in the desert, though when the same car is tested in a large city this may change. Regarding Deep RL the issues are a bit different, though they may not attempt to imitate the training data. RL has the issue of gathering data, as when a supervised network has its data, the training is often relatively quick; an RL algorithm would however often require many tens of thousands if not millions of episodes, to get the same accuracy. Though when the RL network is done training, it

will not necessarily act in the same way a person would.

1.1.3 Transfer Learning

When doing image recognition, a more recent approach to doing this for environments with few example images, is to use what we call Transfer Learning. Transfer learning means that one trains in one environment, and when moving to the next environment, one should use a very small learning rate and then keep training in the new environment. Doing so means that the weights in the network are already initialized in a logical place, giving a higher accuracy or lower loss[11].

1.1.4 Less general methods

There are some less general methods when trying to solve self-driving cars. These less general methods solved the problem of self-driving cars by programming algorithms which solve one task each. Solving one task and using less generalized control methods may solve the problem which the authors of one of these papers want to solve. Though a self-driving car needs to implement many non-general algorithms to solve every problem, this leads to an issue of scalability. As the number of problems increases, the amount of storage and the computational power would have to increase as well, in order to create a truly self-driving car. The scalability issue is also prevalent when considering who will create these algorithms, and tune them to make them work together as one does not necessarily know which algorithm is most important at any time. As an example, let us consider two algorithms controlling a self-driving car, one which uses the GPS and Inertial Measurement Unit (IMU) to control the car; The second algorithm is an algorithm that predicts the lanes. This Inertial Measurement Unit measures the direction and speed with which the car turns, and using this with a GPS allows for a more accurate estimation of the cars location [12]. When the car is driving on a normal road, one might consider the lane detection algorithm to be the best alternative, though issues can arise. If the car is driving off-road, or in a snowstorm with poor visibility,

the lane detection algorithm could predict a lane where there is none, steering the car off the road. At this point, GPS and IMU-based implementations would still function, as these use the GPS signal, and the speed as well as the direction the car is driving to know where to go. The lane detection algorithm could also work correctly, for a new road, which leads to the GPS not having an entry for the new road yet. If the old road ever crosses the new road, the GPS based implementation might then try to get the car back onto the old road.

Even when considering single tasks such as lane detection, there might be issues; one of these is the performance for each algorithm given any environment. Some papers focus on improving the transition between brighter environments, and darker ones [13]. Others focus on improving the lane detection on rural roads [14].

1.1.5 Deep Neural Network Methods

One of the more recent Neural Network methods is called Deep Neural Networks, they are like regular Neural Networks, but just with more layers. Adding more layers allows for pattern recognition on another pattern, resulting in more complex patterns, which then allows for more complex recognition. The Deep Neural Network (DNN) methods have been used in no small degree in image recognition as well as other forms of recognition.

Some of the more recent papers show that DNNs are better than humans at multiple different tasks. One example of image recognition in which AI beats humans is sign recognition [15]. In sign recognition, DNNs recognize signs slightly better than humans. Moving over to a more Reinforcement Learning based example, DNNs also beat humans at certain computer games. In DeepMinds 2013 paper "Playing Atari with Deep Reinforcement Learning" [8] they showed how their Deep Q Network, was able to beat human players at multiple different games such as Breakout, Pong and Enduro. DeepMind then improved upon DQNs by creating what they call Rainbow.

Rainbow took many recent improvements in Deep RL research and tried to implement them to see what created the best Deep RL algorithm [16]. Rainbow included many

different things, such as changing the replay memory, by choosing the memory whose actions give a higher reward more often so that the network learns to act that way, this is called Prioritized replay. On top of Prioritized replay, they use the Double Q learning to address the overestimation, which is prevalent in many DQN networks. The Double Q learning improved by creating the advantage and the value estimation in the same network so that they merely take different paths at the end. The network was also distributed, and then noise was added. The optimization was done for an experience at a time, meaning that the network did not merely take one action and its rewards but a set of actions, and the resulting reward. Training on multiple continuous actions also vastly improved the network. Looking at DeepMinds results, the rainbow algorithm beat out all of the state of the art algorithms at that time. There is an issue with these DNNs, however, namely their computational complexity. As the DNNs are deep, containing multiple layers and millions of variables, this means that they are slower to execute than some shallower methods, this could at worst lead to a worse accuracy as the algorithm is quite a lot slower than other shallower ones.

When explicitly discussing self-driving algorithms, the list of methods used becomes long. The first was a self-driving car, which was called ALVINN or Autonomous Land Vehicle In a Neural Network. ALVINN was a simple network, with a mere two hidden layers, each of which used fully connected neural networks. It performed quite poorly compared to today's standard but predicted the correct action approximately 90% of the time. ALVINN used a lot of electricity and a quite powerful computer at that time [17]. Another notable example is that of Nvidia, which was a Neural Network with five convolutional networks, followed by three fully connected layers. Nvidia's self-driving car received 90% autonomy, which means that the driver corrected the car ten times in 600 seconds [18]. These were both supervised, though there are also some RL-based methods such as Kendall et al. [5], Okuyama et al. [19], and Fayjie et al. [20]. In which they test out DQN networks, in which the network acts and the checks how good the actions were.

1.2 Contributions

The primary contributions of the thesis is creating a system where a road detection algorithm is used as a part of the reward system for an RL based self-driving car. This improved reward system allows for automatic training in a variety of environments, as well as reduces the effort required to retrain the reward system for new environments. For any new environment, only a small dataset is required to train the road detection algorithm. In a real world scenario, systems like this could allow for the successful training of autonomous RL cars without requiring the use of a human as a supervisor. Moreover as part of this work, PPO has been tested alongside DDPG and DQNs to get a conclusive answer to two major questions: whether or not road detection helps improve the RL algorithms If yes, how much does a road detection algorithms impact the performance of RL algorithms Major findings of this work have also been reported as part of the recently accepted conference paper: "Road Detection for Reinforcement learning based autonomous cars"

2 Hypothesis

Using road detection with the RL algorithms during their training will increase the reward gained when testing for a majority of the algorithms, in either environment.

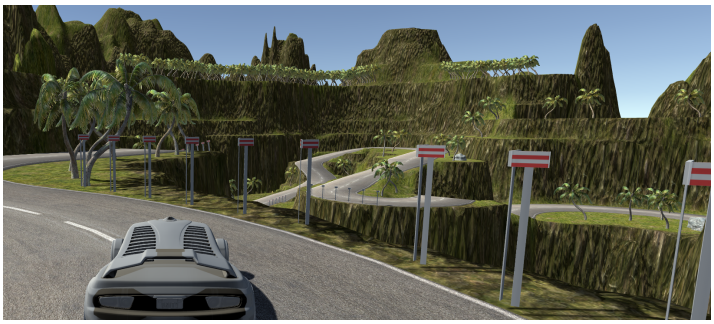
- Using road detection with the RL algorithm will decrease the time taken to train the network.
- Using a larger RL network will improve the reward for each episode.
- Using a larger RL network will increase the time it takes to train an RL network.

3 Approach

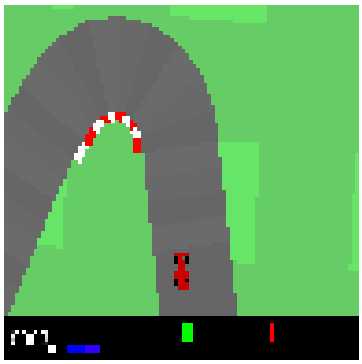
This section introduces the approach to verify the added benefit of road detection in reinforcement learning-based cars. We first introduce the environments, and then follow with explaining the reinforcement learning-based AI-system. We continue by explaining the road detection models used in this thesis.

3.1 Environments

During simulations, the environment sends its reward to the road-detection algorithm along with the image of the environment. The road-detection algorithm then uses this image to check whether or not the car is off the road. If the car is on the road, the reward stays the same. However, when the car was off the road, the reward will decrease, which results in a penalty to the Reinforcement Learning (RL) Algorithm for the actions leading to the car going off-road. Over time this should make the car drive more on the road as it would be penalized for not doing so, thereby learning to drive more quickly.



(a) Unity



(b) Gym

Figure 1: 1a shows the Unity environments complexity, while 1b shows the Gym environments complexity

To test the effect of a road-detection algorithm, we use two simulation environments: (1) OpenAI's Gym environment "CarRacing-v0" and (2) Udacities self-driving Unity environment. These environments are quite different in complexity. The Gym environment is a 2D top-view driving game with no hills or greenery other than the grass off the road. The Udacity environment is a 3D simulation that has mountains, water, and greenery. A consequence of applying to the Gym environment, the task is to keep the car on black (i.e., on the road tiles). The Udacity environment has multiple different colors for its surroundings, including hills that are right next to the road. The complex Unity environment makes it even more difficult as even when the vehicle is on the road; there could be colors other than black or white in the field-of-view. The environment does not use a top-down view, which means that the car's location on the road is implied instead of being explicitly shown.

Udacity's Unity environment was a complex one, with good and bad implementation choices. One of the issues with using Udacitys environment was the lack of reward. For Unity to give a reward to the agent, a system was created. This system took all of the road tiles in the Unity environment and listed them in order of occurrence. Meaning that the road tile that occurred first was the first checkpoint, then the next step was to go from the first one to the next and so on. When the car crossed a checkpoint, it received a reward, which was then used to give feedback to the agent of how well it was doing. Each of the straight checkpoints is approximately a distance of five units from the front to the back, while the non-straight checkpoint varies in length, most of which are longer than five units.

Figure 2: The orange highlights are the different checkpoints

A divider was also added into the environment to separate two roads, which are side by side. The divider meant that the car did not drive from the start to the middle of the map.

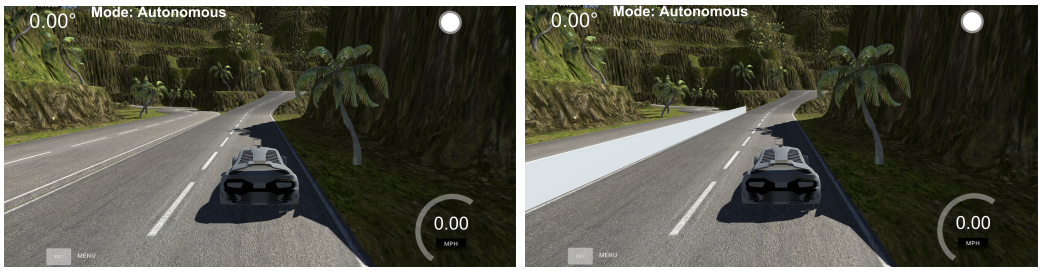


Figure 3: Shows the Unity environment with and without a divider

After this, there needed to be a way to tell the car that it was off the road. In order to do this, all of the road tiles were given a tag: road. If one of the wheels did not have the "road" tag below it, then that means that one of the tires was off the road. This system allowed us to train a road detection system quite quickly in this environment. Lastly, the car needed to reset, every time it got stuck, or was off the road for too long. To reset the car, a script counted how many times the car was off-road, to account for being off-road for too long. In order to check if the car got stuck, a script checked the location of the car 75 actions ago, and if the location was the same for the current action and the location from 75 actions ago, then the car is reset and given a negative reward.

Together these systems allowed us to give a reward when the car kept on driving and penalty for getting stuck or driving off the road.

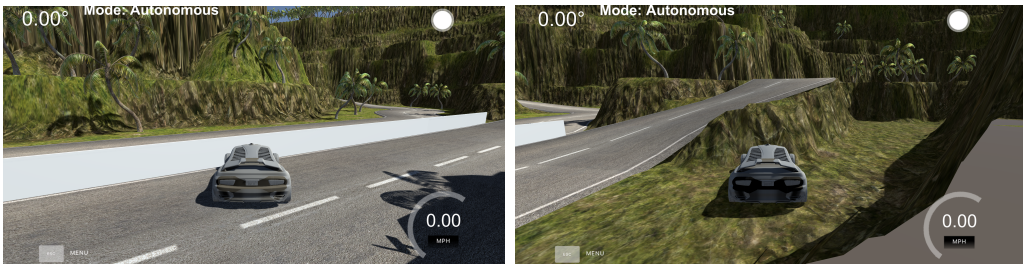


Figure 4: Shows the car being stuck while being on the road, or off the road

Figure 5: Shows the car being reset

As all of the algorithms needed to interact with both of the environments, this led to a small issue. This issue is that of continuous versus discrete action spaces. Continuous actions refer to when the predicted action is a floating-point number, while the Discrete actions are integer predictions. The difference between discrete and continuous actions is essential. The reason for this importance is that the DQN network can only interact with a discrete action space, while Policy optimizers such as PPO and DDPG can interact with a continuous action space. As DQNs cannot interact with continuous action spaces, this required a translation of the discrete actions from the DQN to continuous actions. In order to do so first, we needed to know what was most important, as the Gym environment was quite different from the Unity environment.

The Unity environment had a max speed, meaning that the accelerator was not an issue, as this could just be set to the max acceleration, leading the car to merely controlling the steering wheel. The angle that the steering was critical in this environment, as the car is barely smaller than the lanes, leading the car to have a bit more clearance than two times the width of the car. The Gym environment was different. It lacked any form of max speed, which resulted in the acceleration being an essential factor for the environment, as well as the steering. As the Gym environments road was approximately four times larger than the car, this meant that the car did not necessarily need a wide variety of steering angles. The Gym environment, therefore, had three speeds and two angles for each side. On top of this, there was a break action, and a drive straight action resulting in thirteen actions.

3.2 AI system

The AI system consists of several components. It has a road-detection algorithm, an autonomous driving algorithm, and a way of penalizing the autonomous driving algorithm when it goes off the road. Moreover, as the environments are different, this needs to be handled in distinct ways for the different environments. Handling different environments is done by creating an environment script that takes in the image and reward of either simulation. Given an image, the road-detection algorithm checks whether the car is off or on the road. In the cases where the car is off the road, it decreases the reward.

As the road-detection algorithm has to run beside the RL algorithm, it needs to be rather fast. The faster the road-detection algorithm is, the less it impacts the training time of the RL algorithm. When testing the car in a real-world environment, a fast road-detection algorithm means that the car can react quicker when training it, giving more granularity in its control.

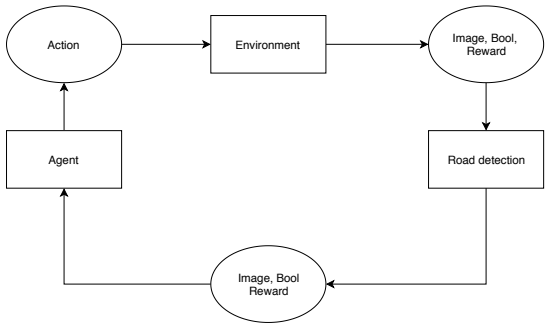


Figure 6: A flowchart of how the system used for training works

3.2.1 Supervised Learning

For the road-detection algorithm, we used a supervised model. The reason for this was that the supervised model would be able to train on millions of images in a few days, then be used to imitate the behavior from these images. Imitating others' behavior is quicker than training an RL algorithm, as they need to act, and each action has a probability of being correct. Given that we are choosing a supervised model, and using it for image recognition, we can then choose amongst the state of the art image recognition algorithms.

We do have one constraint, which is to choose an algorithm that has an impact that is as small as possible on the autonomous algorithms' ability to react as possible. As the supervised learning algorithm is merely helping to train the RL algorithm, it is not involved in the driving of the car. Now that we have these constraints, we can remove some of the more powerful, but larger and computationally complex models from the mix, such as Google's Inception networks, ResNet, Noisy Student, and GPIPE. Leaving us with models such as MobileNet, MNASNet, and FPNas, among these, MobileNet is a quicker network than the others, while still achieving high accuracy [21][22][23][24][25].

3.2.2 Reinforcement Learning

The autonomous algorithm has one major requirement, which is to receive an average reward, which is as high as possible, given the timeline for the training. The reason an RL algorithm is used instead of a supervised algorithm is due to the issue of data collection.

It can take many hundreds of hours to collect enough data for the algorithm to be able to drive nearly correct. As well as the issue of data collection, there is also the issue of imitation. Supervised learning learns to drive in the same way as the data it is trained

on, meaning that if humans get distracted and make the wrong move, the algorithm might do the same given enough samples. Essentially a supervised model needs near-perfect data, as it will then be able to imitate this perfectly. A Reinforcement Learning Algorithm is a bit different, as it performs actions, and given that it performs actions, it checks how good this action was. Meaning that it will primarily start with performing random actions, then after a while, it starts performing less and less random actions.

RL algorithms are, in general, now able to outperform humans in multiple tasks, which could indicate that given enough time to train, it will do the same for driving [8]. Though collisions are a significant issue, given that the cars are training using RL algorithms, as RL algorithms will likely crash many times before becoming proficient at driving.

In a real-world scenario, the risk of collisions is alleviated using a road-detection algorithm. The other algorithms consider the distance the vehicle is from another object. When the distance between the vehicle and other objects gets too small, this is considered a collision into the object, stopping the vehicle by overriding the RL's actions. When this is implemented, the cost of training the vehicles becomes less of an issue, resulting in the opportunity of training an RL based autonomous algorithm. For simulations, however, the road-detection algorithm adds a negative reward to the reward for that timestep to indicate that this is an unfavorable location to be.

The different versions considered for the RL algorithm, are, Deep Q Networks (DQN) by DeepMind [8], Proximal Policy Optimization (PPO) from OpenAI [9], Trust Region Policy Optimization (TRPO) by UC Berkley [26] and Deep Deterministic Policy Gradients (DDPG) from Google [10]. These all perform actions in different ways and have different positives and negatives. As for DQN, it does not act in continuous action spaces. Though the PPO, DDPG, and TRPO will act in continuous action spaces. The performance of DDPG and PPO for this environment is not known. When introducing PPO, however, OpenAI said this about PPO: "These methods have the stability and reliability of trust-region methods but are much simpler to implement..." referring to PPO having many of the benefits of TRPO, though being much simpler to implement. As PPO has many of the same benefits while being "much simpler to implement", PPO is chosen instead of TRPO [9].

3.3 Road Detection Models

When training the RL algorithm, several factors need to be considered, such as how big the network should be to train the vehicle as quickly as possible and still allow for a collision rate, which is as low as possible when training the model. Therefore the choice of model complexity is an important one, should the RL algorithm be a large one, which can perform flawlessly in the environment, but which is slow. Alternatively, should the RL algorithm be a faster algorithm, which may be able to perform more actions in the time-frame, which could allow it to stop a collision as it can predict the imminent collision between the bigger models' actions and stopping in time. So a bigger model will have the ability to generalize more but will run slower, giving it less time to react. Therefore, we test three different models.

3.3.1 Small RL algorithm

We start off testing the smallest implementation of each RL algorithm, a 3-layer implementation of fully connected layers. The time it takes to run through 1000 actions in the Gym environment is approximately 12 seconds when not using the road-detection algorithm. When using the road-detection algorithm, however, this increases to 24 seconds. This is significantly longer, leading to the issue of speed while training. Meaning, does the road-detection algorithm help the training enough to train the network to stay on the road faster than when using not using a road-detection algorithm.

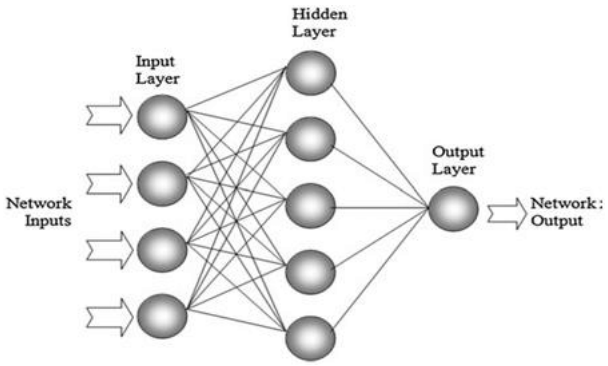


Figure 7: An overview of the small three-layer RL architecture

3.3.2 MobileNet RL algorithm

The smallest algorithm was an implementation with a mere three layers; this is good for speed but leads to less generality. A more general yet quite quick algorithm is MobileNet [27]. MobileNet is created to run on mobile devices, which means that the algorithm is quite quick, though it has more parameters that a simple, fully connected network. A bigger network leads to a higher generality while losing some of the speed of the smaller networks. Though the performance of MobileNet is state of the art, achieving 92.8% on ImageNet, which is lower than state of the art, but still quite reasonable when considering the speed at which it does so [27][21].

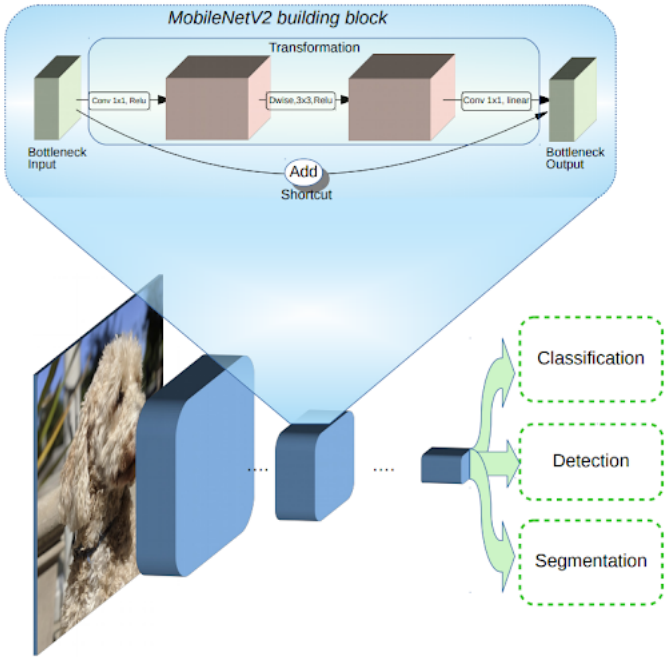


Figure 8: An overview of the MobileNet architecture

3.3.3 Large RL algorithm

Amongst the larger networks, we check the impact of a network, which is state of the art, yet not as large as the absolute best networks, namely EfficientNet-B5 [28]. It has 30 million parameters, which is about six times more than MobileNet while being made to be rather efficient in its computation. A large network leads to more generality than MobileNet, though a longer training time as it can perform fewer actions per second. The questions then become how well it performs compared to the quicker algorithms, is it able to overcome the issue of its speed compared to the other algorithms, with more generality.

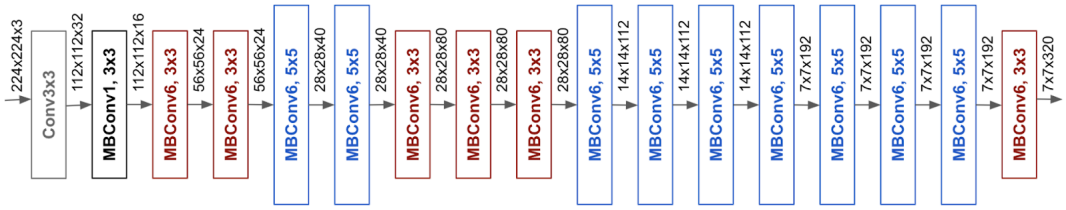


Figure 9: An overview of the EfficientNet architecture

4 Experiments

The experiments show what improvements are there for each algorithm when using a road detection algorithm to give a penalty for driving off the road. The improvement is shown using the reward given by the environment; this includes the penalty given by the environment when the car is off-road. In table 2, the reward received for each of the algorithms is shown both when using the road detection algorithm and when not using it. Both the PPO and the DDPG show results after 12700 episodes when using road detection and 17400 episodes when they do not use a road detection algorithm. The results for the DQN is after 1400 episodes when it uses a road detection algorithm, and 2100 episodes when not using the road detection algorithm.

	With road detection	Without road detection
PPO	-54.4	-55.3
DDPG	-19.9	-23.7
DQN	-38.2	-77.1

Table 2: The average result of the 100 last episodes for each algorithm in the gym environment

Referring to table 3, we see the accuracy increasing for the PPO when it uses the road detection algorithm. The PPO was trained for 15000 episodes, both when it used road detection and when it did not use road detection. Though the DQN was run for 5000 episodes with the road detection, while not being run without road detection.

Other than these results, there are also the results from the MobileNet road detection. The MobileNet algorithm, trained using perfect data in the Unity environment, in which all of the photos were gathered automatically while running the algorithm. While in the Gym environment, these needed to be hand labeled, and there might therefore be a small number of photos which have been mislabeled. The labeled photos were then used to train the algorithm to work for each environment. Once the training was

	With road detection	Without road detection
PPO	10.8	7.5
DQN	0.363	N/A

Table 3: The average result of the 100 last episodes for each algorithms in the Unity environment

completed, transfer learning was used to check how much of an improvement transfer learning made.

Referring to the road detection papers results, table 4 shows the accuracy for the MobileNet algorithm when it is trained in the different environments and tested in both. The table shows the MobileNet algorithms results with and without transfer learning, and the K-means results when the K-means is trained and tested in the same environment.

Train \ Test	Unity	Gym
RD Unity	98.26	51.48
RD Gym	43.05	96.70
RD Unity ⇒ Gym	49.69	97.98
RD Gym ⇒ Unity	99.41	54.09
K-means	93.15	57.05

Table 4: Accuracy's for two different environments, RD is the road detection algorithm, and the arrow represents the transfer learning from one environment to the other. The K-means was both trained and tested in the same environment.

5 Discussion

The results shown are auspicious as every algorithm which had trained for more than ten thousand episodes, showed improvement in their performance when using the road detection algorithm to give a penalty for driving off the road. The reasons the DQN ran for so few episodes compared to the other algorithms are a couple. The first reason is that implementing the DQN network for either environment required much debugging, as it would not run in continuous action space. The Gym environment also did not work well with the DQN network, as OpenAI's own baseline DQN model, does not run in their CarRacing-v0 environment. The Gym environment also returned the data type Long when the reward was 0, and this caused a data type error, which then crashed the program.

Udacity's Unity environment ran as fast as the CPU was able to render it; this led to an issue where the car would drift into mountain walls or the divider, which got the car stuck between actions. This meant that the algorithms could not run on the servers; instead, they were run on my computer, which though it is a powerful 16 core computer, could not run two instances at the same time. The update rate of the environment was also severely impacted by other people running their programs on the university servers.

The reason the DQN network ran slowly in the Unity environment was due to the time it took to update the network after the end of an episode. After the episode was done, the unity environment reset the car and continued the last action until it received the next action from the actor (DQN network). This led to the Unity environment getting stuck while it updated the DQN model; it, therefore, was not updated every episode. Not updating the model every episode led to the model being able to train, though this also slowed down the training of the model significantly.

Regarding the reward increase for the DQN network in the Gym environment, this improvement is significant; however, the car is simply drifting to one side, instead of attempting to drive the car inside of the lane. The DQN without the road detection chose to turn the car as much to the other side as possible, this means that either cases do not seem to have learned anything compared to the DDPG and PPO algorithm.

6 Future Work

In order to get a full overview of how the road detection system improves the performance of the autonomous RL algorithms, more testing is needed. One of the needs is to see the effect of a road detection system on larger RL algorithms. As well as this, there is also a need for more extensive testing, testing the algorithms for longer.

In order to do all of this testing, distributed versions of the code should be created, allowing for faster training. To train using distributed learning in the Unity environment, it needs to be rewritten to update for every action, instead of updating as fast as the CPU can.

Implementing these improvements will allow the network to train faster, giving us more results for the DDPG and the DQN. The improvements would also show the impact of the road detection algorithm for RL networks of different sizes.

7 Conclusion

This thesis demonstrates that the introduction of a road detection algorithm improves the performance of the RL algorithms in autonomous driving. However, the improvement potential varies between environments and algorithms. Most notably, when testing using the PPO algorithm in the Unity environment, the RL-algorithm drives approximately seven distance units further before being reset when it uses road detection than when it does not. This increase can be seen to different degrees for both the environment and all of the tested algorithms.

The performance of the RL algorithm is improved both per episode and given the same amount of training time. However, the introduction of road detection made the RL-algorithm take longer to perform one action.

There is still more testing to be done to check how much better the algorithm performs after more extended experiments, e.g., a few hundred thousands of episodes, though the initial results are promising. With more time, testing the DQN network to a more significant extent would show how well DQN performs, as well as this more testing with larger RL models would show the importance of generality versus speed.

The hypothesis is correct in that the road detection algorithm increases the reward that the RL algorithms receive, though there is still some future work to be done to show its applicability in real-life settings conclusively.

References

[1] WHO, "Road safety." https://www.who.int/news-room/facts-in-pictures/detail/road-safety, Des. 2018.

[2] T. Inc., "Vehicle safety report." https://www.tesla.com/no_NO/VehicleSafetyReport?redirect=no, Aug. 2019.

[3] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[4] H. Kong, J.-Y. Audibert, and J. Ponce, "Vanishing point detection for road detection," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 96–103, IEEE, 2009.

[5] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," *CoRR*, vol. abs/1807.00412, 2018.

[6] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

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

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," *CoRR*, vol. abs/1808.01974, 2018.

[12] A. B. Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine vision and applications*, vol. 25, no. 3, pp. 727–745, 2014.

[13] Y.-M. Chan, S.-S. Huang, L.-C. Fu, and P.-Y. Hsiao, "Vehicle detection under various lighting conditions by incorporating particle filter," in *2007 IEEE Intelligent Transportation Systems Conference*, pp. 534–539, IEEE, 2007.

[14] J. Choi, J. Lee, D. Kim, G. Soprani, P. Cerri, A. Broggi, and K. Yi, "Environment-detection-and-mapping algorithm for autonomous driving in rural or off-road environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 974–982, 2012.

[15] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks.," in *IJCNN*, pp. 2809–2813, 2011.

[16] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[17] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, pp. 305–313, 1989.

[18] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[19] T. Okuyama, T. Gonsalves, and J. Upadhay, "Autonomous driving system based on deep q learnig," in *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, pp. 201–205, March 2018.

[20] A. R. Fayjie, S. Hossain, D. Oualid, and D. Lee, "Driverless car: Autonomous driving using deep reinforcement learning in urban environment," in *2018 15th International Conference on Ubiquitous Robots (UR)*, pp. 896–901, June 2018.

[21] J. Cui, P. Chen, R. Li, S. Liu, X. Shen, and J. Jia, "Fast and practical neural architecture search," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[22] P. J. Grother, "Nist special database 19," *Handprinted forms and characters database, National Institute of Standards and Technology*, 1995.

[23] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[24] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[26] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[27] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[28] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

8 Appendix

8.1 Code

Below are the links to the githubs for all of the code used in this thesis:

`https://github.com/marho13/Gym-Master`

`https://github.com/marho13/udacityReinforcement`

`https://github.com/marho13/UnityRL_implementations`

8.2 Accepted Publication

The accepted publication is appended below. The conference paper is called: "Road Detection for Reinforcement learning based autonomous cars" and is accepted by ICISS 2020 (March 19-22) with minor changes. The paper is part of the masters thesis.

Road Detection for Reinforcement learning based autonomous cars

Martin Holen
Centre for Artificial Intelligence Research, University of Agder
4604 Kristiansand, Norway
martin.holen@uia.no

Rupsa Saha
Centre for Artificial Intelligence Research, University of Agder
4604 Kristiansand, Norway
rupsa.saha@uia.no

Morten Goodwin
Centre for Artificial Intelligence Research, University of Agder
4604 Kristiansand, Norway
morten.goodwin@uia.no

Christian W. Omlin
Centre for Artificial Intelligence Research, University of Agder
4604 Kristiansand, Norway
Christian.omlin@uia.no

Knut Eivind Sandsmark
Sopra Steria
4604 Kristiansand, Norway
knut@sandsmark.net

ABSTRACT

Human mistakes in traffic often have terrible consequences. The long-awaited introduction of self-driving vehicles may solve many of the problems with traffic, but much research is still needed before cars are fully autonomous.

In this paper, we propose a new Road Detection algorithm using online supervised learning based on a Neural Network architecture. This algorithm is designed to support a Reinforcement Learning algorithm (for example, the standard Proximal Policy Optimization or PPO) by detecting when the car is in an adverse condition. Specifically, the PPO gets a penalty whenever the virtual automobile gets stuck or drives off the road with any of its four wheels.

Initial experiments show significantly improved results for PPO when using our Road Detection algorithm, as compared to not using any form of Road Detection. In fact, without this detection algorithm, the vehicle often gets into non-terminating loops (for example, driving into the dividers, getting stuck, or driving into a pit).

CCS CONCEPTS

• **Theory of computation → Reinforcement learning**; **Online learning theory**.

ICISS 2020, March 19 - March 22, 2020, Cambridge, UK

ACM ISBN 978-1-4503-7725-6.
https://doi.org/10.1145/3306307.3328180

KEYWORDS

Reinforcement learning, neural networks, Road Detection, Simulation

ACM Reference Format:
Martin Holen, Rupsa Saha, Morten Goodwin, Christian W. Omlin, and Knut Eivind Sandsmark. 2019. Road Detection for Reinforcement learning based autonomous cars. In *International Conference Proceedings Series by ACM*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3306307.3328180

1 INTRODUCTION

Every year nearly 1.25 million people die in traffic, much of it caused by human error. A potential solution to this high death rate is self-driving vehicles, as computers do not suffer from the same drawbacks as human drivers, for example, causing fatal mistakes when tired[15]. Self-driving algorithms are, to some extent, part of modern cars, and data from companies such as Tesla show examples of self-driving cars are already safer than human drivers[9]. These self-driving vehicles are mostly trained using human drivers' data. A notable limitation of this approach is the repetition of human mistakes. One way of counteracting the weakness is through Reinforcement Learning.

Reinforcement Learning (RL) in self-driving cars do not learn using data previously collected from human drivers. Instead, RL algorithms explore and reinforce correct actions and penalize wrong ones (for example, driving off the road). Much research is still needed. In fact, seemingly straight forward problems, such as proper lane detection, is not in place. Further, there is no common understanding of which learning functions to use in any given environment — in conclusion, enabling self-driving in cars if far from a resolved research problem.

Partly autonomous cars, which is a step towards self-driving vehicles, are already an industry standard and affordable for many people. Such cars are available for approximately 45,000 USD which makes them affordable for a relatively large group of people. And given that auto companies now offer vehicles that can perform multiple complex actions such as lane changing to overtake another car, adaptive cruise control, collision avoidance, collision detection, and more, improvements on these systems are even more important[8].

A predefined rule-based system or machine learning-based classification is the basis for most autonomous driving available in the literature. Machine learning-based driving is challenging for companies that do not have hundreds of thousands of cars collecting data. Further, since privacy concerns have become more and more critical in recent years, there is an added difficulty in data collection. Moreover, data of human driving is imperfect as many collisions happen due to human error[15].

Reinforcement Learning (RL) is a machine learning subfield that has received a lot of focus recently. Perhaps the most notable examples are AlphaGo beating the world's best Go player, and AlphaZero beating the best chess AI's[18][19]. The basic principle of this approach is to arrive at a set of appropriate actions via rewards or penalties associated with learned actions. RL is also used for self-driving cars. Here, instead of training on data from human drivers, the RL algorithm typically learns to drive in a simulated environment. In simulations, wrong actions have little consequence. Once the learning is proficient, transfer learning is used to train in a physical environment. The combination of simulation and transfer learning speeds up the training process compared to training only in a physical environment[10].

This paper focuses on a small but crucial sub-problem in the RL self-driving scenario – namely lane detection. We introduce a Road Detection algorithm to support an RL-based self-driving algorithm (for example, the Proximal Policy Optimization or PPO), by informing the RL algorithm when the car is going off the road in multiple different environments.

2 BACKGROUND

In recent years, autonomous vehicles have been the object of a lot of research across industry and academia.

One common approach is to have an RL based self-driving car that uses human interaction as part of the reward function [10]. The setup is so that whenever the vehicle drives off the road, a human trainer stops the car, which the reward function interprets as a penalty. This penalty is, in turn, used to correct the vehicle's actions in the algorithm. Not surprisingly, the training process becomes quite expensive and time-consuming – a human always has to be in the car. Another challenge is that human drivers become tired, which could lead to slow corrections, thereby affecting the efficacy of the reward and penalty system. As a consequence, the system could learn incorrect or unsafe actions that may cause accidents and injuries.

Before the rise of deep learning, most methods for creating self-driving cars were based on using sensor data and manually creating rule-sets for driving behavior. These systems employed various strategies, such as k-means clustering, Dijkstra, perspective transformation. The general actions performed here were mapping, planning, and control, and getting the data from sensors and estimating the state of the car. Many of these methods used computer vision. Cameras or LIDARs were used to get an image of the surrounding area so that the system was able to map it. The process depended mainly on the task which the researchers focused on, such as Road Detection in different lighting conditions using vanishing point detection or Road Detection for rural areas. Such systems, not surprisingly, worked well in those specific conditions but could struggle in a more generalized environment[12][4][1].

In recent years, there has been a broader focus on creating supervised image recognition models whose objective was to imitate the driving of humans. Resulting in not only larger more accurate models, but also computationally effective ones[7]. These supervised models, learn by getting an input predicting an output. This output is checked versus a label, resulting in a difference between the prediction and the label. The model then optimizes to get closer to the labels for any input it is given. The models created to do so have become extremely good at these tasks, even surpassing humans in some of them[3][6].

Supervised systems work quite well in general, but vital issues remain. Mainly, it is challenging to gather a large enough amount of training data collected by experts. A mitigating is the approach carried out by Tesla: every time the driver takes control, it is assumed that the cars' prediction was incorrect. The drawback to this, again, is if the drivers' actions are less than ideal, the vehicle will be trained to drive in that specific manner. Nvidia has also created a simple yet effective imitation based model, based on the earliest Neural Network autonomous vehicle implementation (ALVINN)[16]. The Nvidia self-driving car worked reasonably well, though the model needs to be shown many different scenarios to be able to perform the correct actions, which does not scale well[21][2].

Reinforcement Learning takes an input (s), performs an action (a), and gets a reward (r). A common environment for RL is a Markov decision process (MDP). An MDP contains a state *s*, and action *a*, a probability transition *p*, and gives a reward *r* for the action which leads to the state *s+1*. Given the MDP, the RL algorithm can predict the best action to take.

A common approach is to estimate the reward of each action is Deep-Q networks (DQN). Of which notable examples are [5][14]. For Fayje et. al. the task was to create an autonomous robot which learned to navigate using a Deep-Q network (DQN), whose inputs were a camera and a LIDAR[5]. While Okuyama, Gonsalves et. al. focused on creating a simulation in which the task was to avoid obstacles, this task was solved using a DQN[14]. The estimation of a reward is essential to a DQN along with the verification of the discounted reward, which is collected after performing any action. The discounted reward is a way of saying that all the actions which lead up to a reward were important, ensuring that the network rewards the action completing a goal (for example, a lap in a racing game) but also the other action[13].

Another RL method, also used for self-driving cars, is that of policy gradients, in which the policy predicts which action is the best through an actor. The policy is then later updated based on how good that action was, which is determined by a critic. Promising examples of policy gradients include Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO), of which PPO is said to be "much simpler to implement, more general, and have better sample complexity (empirically)."[17][20][11].

The way all of these methods work is by performing what are random actions at the beginning, then updating the reward for the set of actions the RL algorithm performed. Eventually, the system will learn which actions are beneficial, and in turn, become more and more confident in its operations, making the activities less and less random.

3 APPROACH

A notable drawback of the previously discussed methods is the difficulty in scaling up such systems. The apparent consequence of this is that cars would likely get damaged often in the beginning, and it would be unsafe for humans to stay near the car. Hence, there is a need for a reliable method to detect whether the vehicle is on the road. Through RL, any off-road vehicle should yield a penalty. This would result in a so-called Road Detection algorithm. An algorithm that detects when the car is off the road, other parts of the system can perform actions to get it back onto the road.

There is also the issue of time, as a car is only able to perform up to 10's of actions every second. An improvement of the training speed could be achieved by training in simulations until the car has become proficient enough to be transferred into a real-world scenario. We use transfer learning to move knowledge from one environment into another one.

In our approach, the Road Detection algorithm will be running alongside the Reinforcement learning algorithm (see Figure 3). A challenge with this approach is the size and computational complexity of the Road Detection algorithm. An

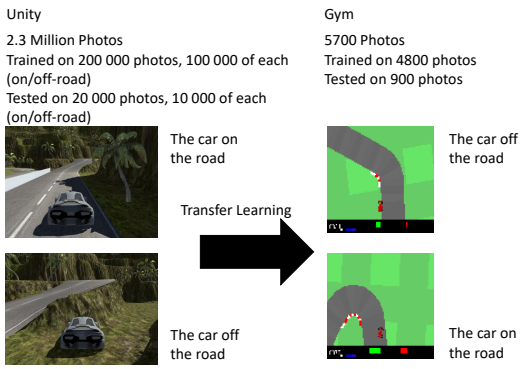


Figure 1: Information about the different environments, and how they are trained. With the Unity photos shown, being a screenshot instead of the image sent to the agent.

algorithm that slows down the RL algorithm would impact how quickly the car can react. It is, therefore, necessary for the Road Detection algorithm to be efficient in terms of both memory and computational power. Thus the choice of the Road Detection algorithm is between algorithms made for mobile devices, the state of the art of which is *MobileNet*[7]. MobileNet which was introduced by Howard et. al. in 2017, is a Neural Network made to be computationally light and thereby usable for mobile devices such as phones. MobileNet uses depth-wise separable convolutions, which reduces the number of variables in the convolutions, as well as the size. It also has two hyperparameters, which have trade-offs between latency and accuracy. These hyperparameters are a width multiplier, which can make the model thinner or thicker as well as a resolution multiplier, which can reduce the representation. The balance of these can be used to create either a faster algorithm or a more accurate (but slower) model. The hyperparameter choice is vital as a loss in accuracy could impact the usefulness of the Road Detection algorithm[7].

For training and experimenting with our proposed Road Detection algorithm, we use an RL-based self-driving car in simulated environments. Using Proximal Policy Optimization (PPO) for driving avoids extensive policy updates. The PPO predicts where the vehicle should drive. Secondly, the Road Detection algorithm is MobileNet, a convolutional neural network that gives feedback to the PPO whenever the car is off the road. MobileNet is quite fast, with only a small impact on the accuracy[7].

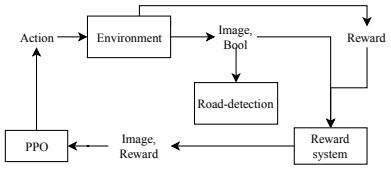


Figure 2: An overview of how the system works during training of the Road Detection Algorithm and the PPO given the Unity environment.

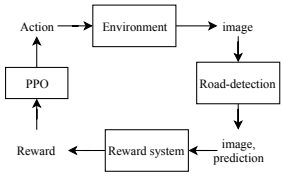


Figure 3: An overview of how the system works when the Road Detection algorithm is trained, and is used to train the PPO.

4 ENVIRONMENTS

Training and testing of the Road Detection algorithm was done with two separate environments: A Unity environment created by Udacity, as well as carracing-v0 from OpenAI's gym. Using different environments means that there is a need to be able to use the same setup for each environment. We achieve this by calculating the reward outside of the Unity environment.

The Udacity environment is a 3D racing game with twists and turns, as well as hills and bridges. When one acts in the environment it feeds both the image and a Boolean value indicating if the car is off the road. The boolean is sent to a reward function, which in turn sends the image and the reward function onward to the PPO and the Road Detection algorithms. This duality enables training of both algorithms simultaneously, which can be seen in figure 2. The images given from the unity environment is not in the standard RGB format, resulting in the images it sends looking a bit different than the gym images, for reference look at figure 4.

Figure 4: Shows the images given from each environment, during the training and testing phase

The Gym environment is a 2D racing game, with some turns. When interacting with this environment it gives the state in the form of an image, along with a reward, and a boolean which says whether the episode is done or not. The RL algorithm interacts with the environment simply by saying environment.step(action), which applies the action to the environment and sends back the aforementioned variables.

During the testing of the algorithm, a similar process is carried out. The environment also sends the image to the Road Detection algorithm, which again is forwarded to the reward function along with the Boolean on-or-off road value. All of this is delivered to the PPO, which uses the information to collect a reward for its driving.

Overall these environments are quite different, as the Udacity environment is a rather complex 3D racing game, while the Gym environment is a rather simple 2D racing game. For reference refer to figure 5.

Figure 5: Shows how the complexity is different for each simulation

5 RESULTS

The Road Detection algorithm needed to be tested for two different goals, to verify that its accuracy was high enough to be used in the reward system, as well as verifying how much it improved the self-driving algorithm. For both the test of the accuracy and the improvement in the PPO's actions, there were two environments; namely Udacities Unity environment, as well as OpenAIs gym environment carracing-v0. This provided evidence that the Road Detection algorithm was suitable for multiple different environments of various complexities.

Train \ Test	Unity	Gym
RD Unity	98.26	51.48
RD Gym	43.05	96.70
RD Unity ⇒ Gym	49.69	97.98
RD Gym ⇒ Unity	99.41	54.09
K-means	93.15	57.05

Table 1: Accuracy's for two different environments (Unity and Gym), RD is the Road Detection, and the arrow represents transfer learning (from x ⇒ y). The K-means was both trained and tested in the environment corresponding to that column.

Table 1 presents the accuracy for applying the method in the Unity and Gym environment. The table shows, not surprisingly, that training in one environment (for example, Unity) and validating in another (for example, Gym) yields a lower accuracy than training and testing in the same environment. Further, by using transfer learning by training in one environment, and continue training in another, this gives even better results. When we train and validate in Gym alone the model gives an accuracy of 96.70, but transfer learning from Unity to Gym increases the accuracy to 97.98. This is likely due to the weights from training in the Unity environment being a better starting point than the initial weights given by simply initiating the model.

When looking at the accuracy for the Unity environment given that the model is trained in the gym environment, the accuracy is rather low at 43.05% accuracy. This is likely due to the color scheme from the Unity environment being quite different from the standard RGB format given in the Gym environment, resulting in the model not functioning correctly, as the weights will only be tuned for the one environment. To see how the Unity environments images look, refer to figure 4.

If the PPO algorithm does better when it uses the Road Detection algorithm, this means that the Road Detection improves training. The validity was verified in the modified Unity environment, which had checkpoints in it, these checkpoints represented how far the car drove.

PPO with Road Detection	4.8 Checkpoints
PPO without Road Detection	3.6 Checkpoints

Table 2: The performance of the PPO in the Unity simulation, with and without the use of the Road Detection algorithm during training.

Table 2 shows the average number of checkpoints the PPO was able to drive past, with and without the Road Detection algorithm when training. This shows how the PPO is able to drive further when using a Road Detection algorithm, than when not using one. Showing that the Road Detection is an improvement to the PPO during training.

6 CONCLUSION

In this paper we create a novel system which gives an automatic feedback, using a state of the art Road Detection algorithm. This feedback system automates the penalty for driving off the road, for a Reinforcement Learning algorithm which can be used in different environments. Allowing for the training of autonomous vehicles in different environments without the need to modify the environments. In turn resulting in less work when training the RL algorithms, while speeding up the training process.

REFERENCES

[1] Álvarez, J. M., Gevers, T., and López, A. M. Road detection by one-class color classification: Dataset and experiments. *CoRR abs/1412.3506* (2014).

[2] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. End to end learning for self-driving cars. *CoRR abs/1604.07316* (2016).

[3] Ciresan, D. C., Meier, U., and Schmidhuber, J. Multi-column deep neural networks for image classification. *CoRR abs/1202.2745* (2012).

[4] Dahlkamp, H., Kaehler, A., Stavens, D., Thrun, S., and Bradski, G. R. Self-supervised monocular road detection in desert terrain. In *Robotics: science and systems* (2006), vol. 38, Philadelphia.

[5] Fayjie, A. R., Hossain, S., Oualid, D., and Lee, D. Driverless car: Autonomous driving using deep reinforcement learning in urban environment. In *2018 15th International Conference on Ubiquitous Robots (UR)* (June 2018), pp. 896–901.

[6] Ho-Phuoc, T. CIFAR10 to compare visual recognition performance between deep neural networks and humans. *CoRR abs/1811.07270* (2018).

[7] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR abs/1704.04861* (2017).

[8] Inc, T. Introducing navigate on autopilot, 2018.

[9] Inc, T. Tesla vehicle safety report, 2019.

[10] Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J., Lam, V., Bewley, A., and Shah, A. Learning to drive in a day. *CoRR abs/1807.00412* (2018).

[11] Lazar, D. A., Biyik, E., Sadigh, D., and Pedarsani, R. Learning how to dynamically route autonomous vehicles on shared roads, 2019.

[12] Miksik, O. Rapid vanishing point estimation for general road detection. In *2012 IEEE International Conference on Robotics and Automation* (May 2012), pp. 4844–4849.

[13] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR abs/1312.5602* (2013).

[14] Okuyama, T., Gonsalves, T., and Upadhay, J. Autonomous driving system based on deep q learnig. In *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)* (March 2018), pp. 201–205.

[15] Organization, W. H. Global status report on road safety 2018, 2018.

[16] Pomerleau, D. A. 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.

[17] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR abs/1707.06347* (2017).

[18] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science 362*, 6419 (2018), 1140–1144.

[19] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T. P., Simonyan, K., and Hassabis, D. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR abs/1712.01815* (2017).

[20] Vinitsky, E., Kreidieh, A., Flem, L. L., Kheterpal, N., Jang, K., Wu, C., Wu, F., Liaw, R., Liang, E., and Bayen, A. M. Benchmarks for reinforcement learning in mixed-autonomy traffic. In *Proceedings of The 2nd Conference on Robot Learning* (29–31 Oct 2018), A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87 of *Proceedings of*

Machine Learning Research, PMLR, pp. 399–409.

[21] Xiao, T., Xia, T., Yang, Y., Huang, C., and Wang, X. Learning from massive noisy labeled data for image classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015).