Universiteit Leiden
The Netherlands

Opleiding Informatica

Sim-to-Real autonomous driving in CARLA using Image Translation and Deep Deterministic Policy Gradient

Jochem Ram

Supervisors:
Dr. E.M. Bakker
Prof. dr. Michael S. Lew

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

30/01/2022

Abstract

Reinforcement learning is a relatively new approach to train an agent to drive on public roads. Training an autonomous driving agent in a live environment unfortunately is challenging, since this would by definition involve making mistakes on public roads and potentially causing accidents. An option is to train the agent in a simulation, but this introduces challenges of its own, notably the simulation-to-reality gap.

This project proposes a framework for training autonomous driving agents consisting of the CARLA urban driving simulator, Deep Deterministic Policy Gradient algorithm and image translation, with the goal of training an agent in a simulation while simultaneously reducing this sim-to-real gap. In this framework, the frames rendered by the simulator are translated to look more similar to real-world images while preserving the important features. We train two model variations: one where conventional convolutional layers are used for feature extraction, and one where the InceptionV3 [1] neural network is used. By measuring the accuracy of these models on a real-life driving dataset, the performance of the models trained on this framework with image translation will be compared to the performance of identical models that were trained without image translation. The accuracy of our model will also be compared to the accuracy of models created in earlier works, which were trained using the Asynchronous Advantage Actor Critic algorithm [2] and the TORCS driving simulator [3]. Preliminary results show that the image translation models return useful images that retain the overall structure of the virtual image. While the driving models as they were trained in this paper have not developed particularly useful policies, the models instantiated with InceptionV3 [1] do show some potential and could be a good starting point for further research.

Contents

1 Introduction

Autonomous driving is a concept that promises to make cars even safer and easier to use than they are today. Many cars already help the driver in some shape or form: according to the American Automobile Association, 97.2 percent of vehicle models available for purchase in the United States in May 2018 contained some form of driver assistance [4]. Companies like Nvidia, Google and Tesla take this a step further and are actively trying to develop fully autonomous driving systems that require no user input, other than setting the desired destination.

In fully autonomous driving, an autonomous driving agent is responsible for taking appropriate actions in given situations. This agent will perceive the state of the environment it is in through the sensors present on the vehicle. The agent will then combine this sensor input with a previously learned policy. Using this policy, the agent tries to predict the best course of action for that given state. It will then use the actuators present on the vehicle to execute the desired action. This paper is concerned with teaching an agent a driving policy for urban and highway settings.

There are roughly four ways to teach a policy to a computer. First of all, one can explicitly program what action should be taken for every possible state. This can be applied to many computational problems, but in autonomous driving there are so many possible states that this approach is virtually unworkable. Unsupervised learning can be used to train on unlabeled datasets to estimate certain properties about the data, which is useful for data analysis tasks but is not a very viable approach to autonomous driving. Two approaches remain: supervised learning and reinforcement learning (RL). In supervised learning an agent is trained on a labeled dataset, so the desired output corresponding to inputs is known. During training, the agent makes predictions on the inputs, and the true outputs are used to correct the agent depending on its predicted outputs. with RL, no labeled datasets are used for training. Instead, an agent is given an environment to interact with. The agent's actions are evaluated with a reward function, where desirable behavior receives higher rewards. The goal of the agent is to maximize this reward over its lifetime. Both supervised and reinforcement learning have their advantages and disadvantages when autonomous driving is concerned.

First, we will discuss supervised learning. Because supervised learning uses a dataset, it is easier to control what the agent learns by altering the contents of the dataset. Supervised learning may have certain limitations: the policy learned will only reflect the data available from the dataset. If the dataset contains mostly data about driving in a straight line on an empty highway, one can expect the agent to become quite skilled at driving on an empty highway. However, this characteristic can also be a disadvantage: the agent cannot learn to handle situations that the training dataset does not contain. Using the highway dataset from before, the agent would not be able to learn how to drive in a city environment. Similarly, an agent would not know how to act in uncommon situations, such as accidents, if these are not part of the training data. It is challenging to incorporate all scenarios the agent might encounter during deployment into the training dataset.

Reinforcement learning takes a different approach, where no dataset is involved at all. Instead, the agent has to explore its environment on its own. As long as the environment contains everything the agent might run into during deployment, it can learn everything it needs to, given enough time. Accidents and mistakes are also part of the experiences the agent will gather. This aspect of exploration is problematic when training the agent in a live environment, as this could lead to dangerous situations for other traffic participants.

A proposed solution is training a reinforcement learning agent in a simulation instead of training on real roads. After training, the agent would be deployed in the real world. With this

approach, there is no risk of physical damage during training. An added benefit is that the environment could be simulated at a faster rate than reality, given enough computing power. This would allow the agent to consume an even larger volume of training data.

Of course, training in a simulator is not without challenges of its own. The challenge this paper tries to tackle has to do with the simulation-to-reality gap. Even though computer graphics have improved significantly in realism over the past few decades, the difference between reality and simulation is still noticeable.

This paper explores whether real-world performance can be improved by making the virtual image look more similar to the real world. To our knowledge, this is the first attempt to train an end-to-end driving model using DDPG with visual data processed by image translation as input. It also builds on the work of Pan et al. [5] by using a simulator that simulates an urban driving environment, rather than a racetrack environment.

In Section 2 previous work will be introduced that serves as a basis for this project. After that, we will explain some fundamentals and terminology that will be used in the rest of the paper in the Section 3. A description of the framework proposed in this paper is given in 4. The experimental setup is presented in Section 5, and the results can be found in Section 6. Finally, we present the conclusions and discussions on this project in Section 7.

2 Related Work

Until a decade ago, most methods to learn driving policies relied on hand-engineered algorithms or imitation learning, but in the past few years promising progress has been made applying reinforcement learning to vehicle navigation [6].

Supervised learning is one of the earlier approaches for training an autonomous driving agent. Perhaps the earliest example of a neural network trained for this purpose is ALVINN [7], where a network consisting of 3 fully-connected layers used image and laser rangefinder input to follow a road. More recently, [8] and [9] have leveraged pre-recorded labeled driving footage to train an end-to-end driving agent, a variation on supervised learning known as imitation learning.

This project uses a reinforcement learning algorithm that does not use a labeled dataset, but learns by exploring its environment. As mentioned earlier, the training often takes place in a simulation. [10] is an early example of this, where the researchers created their model for a vehicle and trained it to make decisions in a highway environment. In [11] reinforcement learning is applied to teach a car an end-to-end driving policy in the TORCS [3] simulator using the Deep Deterministic Policy Gradient (DDPG) [6] algorithm.

2.1 Image Translation

A major issue when training an agent in a simulator for real-world deployment is the simulation-to-reality gap. Observations in a simulated environment are different from observations made in reality, and as a result agents trained in simulations might perform well in those simulations, but worse in reality. Domain randomization is proposed as a potential solution for this gap in [12]. Domain randomization randomizes the visual output of the simulator, resulting in the agent training on different environment variations. Reality would then appear as just another random variation of the simulator to the agent.

Efforts to bridge the simulation-to-reality gap have also been made with image-to-image translation. For example, the simulated image can be translated into a more realistic-looking image before training the agent [5]. An even more recent proposed method is harmonizing the

feature space of the simulated and real images and training the agent on the feature space [13][9].

Image translation is an important component of these simulation-to-reality approaches. An early approach to image translation can be found in Hertzmann et al.'s work on image analogies [14], where the authors create image filter effects using a filtered and unfiltered image pair as input. Similarly, Shih et al. [15] propose a method to synthesize outdoor photos at different times of day from two source photos: one of the landscape and one of the desired time of day. More recently, Isola et al. developed a framework to perform a variety of image translation tasks [16]. A generative adversarial network (GAN) is trained on example image pairs, with the goal of generating images that cannot be distinguished from real images. Because Pix2Pix is not limited to one type of transformation, and because Isola et al.[16] already demonstrated the performance of Pix2Pix in generating realistic urban driving images when trained on the CityScapes [17] dataset, we use Pix2Pix as our image translation method.

2.2 Learning in a Simulator

The concept of image translation has already been used to bridge the sim-to-real gap of autonomous agents. The work by Pan et al. [5] is to our knowledge the earliest example where proper image translation was used during training, rather than a form of domain randomization. It is also the baseline for our paper. In [5], the agent is trained in the TORCS simulator [3] using the asynchronous advantage actor-critic (A3C) algorithm presented in [2].

The aim of this paper is to compare the performance of the approach in [5] to the performance of an agent trained in a more versatile simulator, as TORCS simulates a relatively simple racetrack environment. The CARLA [18] simulator was chosen, because it emulates a driving environment more like what would be encountered on public roads. It also has an extensive API that provides access to environment and vehicle data that we can use in our reward function.

This paper also employs a different algorithm than the baseline from [5]. Instead of A3C with discrete actions, Deep Deterministic Policy Gradient (DDPG) [6] with continuous actions is used. DDPG is similar to the established Deep-Q learning introduced by Mnih et al.[19] in that it uses a network to measure the 'goodness' of actions in certain situations. Unlike Deep-Q learning, however, DDPG operates in continuous action spaces.

3 Background

At the heart of any reinforcement learning process is the training algorithm, which in our case is Deep Deterministic Policy Gradient. This algorithm will be used to teach an agent to drive in urban and highway environments. The environments for this project will be provided by CARLA, a free and open-source driving simulator. The observed environments also need to be translated into a more realistic image, to attempt to reduce the sim-to-real gap mentioned in Section 2.1 for autonomous driving application. The Pix2Pix image translation framework [16] will be used to address this challenge.

3.1 Deep Deterministic Policy Gradient

The Deep Deterministic Policy Gradient Algorithm, or DDPG for short, was developed in 2015 by [6]. They combined the then-established Deep Q-learning with concepts from actor-critic models, specifically the Deterministic Policy Gradient (DPG) algorithm developed by Silver et al. [20]. The full DDPG algorithm from that paper can be found in Algorithm 1.

In Deep Reinforcement Learning, an agent interacts with an environment E. Every discrete time-step t the agent performs an action a that takes the environment from state s_t to state s_{t+1}. The agent also receives a reward r_t, which is bigger the more desirable the new state is. The action to be taken at any given state is determined by a policy π. The policy maps states to a probability distribution over actions:

$$\mathcal{S} \rightarrow \mathcal{P}(\mathcal{A}) \tag{Eq. 1}$$

The goal of the policy is to maximize the expected total reward over a complete episode.

The environment and interactions of the agent with the environment can be modeled as a Markov Decision Process (MDP). An MDP consists of:

- The state space $\mathcal{S}$ containing all possible states.
- The action space $\mathcal{A}$ containing all possible actions.
- $P_a(s, s')$, the probability that performing action a in state s at time t will lead to state s' at $t + 1$.
- The reward function $R_a(s, s')$ which determines the immediate reward for performing action a in s to reach s'.

The expected total reward over an episode $V(s)_\pi$ if a certain policy π is followed starting in state s is given by the state-value function, defined as:

$$V(s)_\pi = \mathbb{E}[R] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi\right] \tag{Eq. 2}$$

Here, $\mathbb{E}$ is the expected value and R is the state return defined as the sum of discounted rewards and:

$$R = \sum_{i=t}^{T} \gamma^i r_i \tag{Eq. 3}$$

where γ is the discount rate that emphasizes the future rewards less or more, depending on its value.

Similarly, we can define an action-value function (otherwise known as a Q-function or Q-value), which assigns a value to an action a taken in state s under policy π:

$$Q^\pi(s, a) = \mathbb{E}\left[R | s_t = s, a_t = a, \pi\right] \tag{Eq. 4}$$

It is useful to define the Q-function whose predictions always result in the highest possible total reward as the optimal Q-function as $Q*$:

$$Q^*(s, a) = \mathbb{E}\left[r + y \max_{a'} Q(s', a')\right] \tag{Eq. 5}$$

In some environments it is possible to keep track of all the state-values and Q-values (action-values) possible in the environment, e.g. in a table. However, this is not an option in an urban driving environment, as the amount of possible states is infinite. Instead, we can estimate the Q-values from previous experiences. This is the job of the critic network in DDPG. This network receives a state-action pair and outputs the estimated Q-value for that pair.

The critic network Q contains weights θ^Q, which can be adjusted to train the network. Just as in Q-learning [19], these parameters are adjusted to minimize the mean-squared error in the Bellman equation:

$$L_t(\theta^Q) = \mathbb{E}\left[y_t - Q(s, a|\theta^Q))^2\right] \quad \text{(Eq. 6)}$$

Here y_t denotes the target values, denoted by

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) \quad \text{(Eq. 7)}$$

During training a gradient is calculated over the loss, and using these gradients the critic network is updated through an optimization algorithm such as stochastic gradient descent.

According to the paper in which the DDPG algorithm was presented, using the weights θ^Q to determine the target value in Equation Eq. 7 to update the same weights with Equation Eq. 6 will often prove to be unstable [6]. To improve stability, a copy of the critic network is used: the target network $Q'(s, a|\theta^{Q'})$. This network is slowly updated to track the actual network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$, with τ as the target learning rate. This network is used to calculate the target values in Equation Eq. 7.

In DDPG, the actions taken are generated by an actor network $\mu(s|\theta^\mu)$. This network receives the current state as input and maps this to an action output. It is updated according to the policy gradient as described in the paper on DPG by silver et al. [20]:

$$\nabla_{\theta^\mu} J = \mathbb{E}\left[\nabla_a Q(s, a|\theta^Q)\nabla_{\theta^\mu}\mu(s|\theta^\mu)\right] \quad \text{(Eq. 8)}$$

The networks are not trained on the experience of the last time-step. An experience replay buffer is used instead, which stores a large but finite number of experiences. Using a replay buffer ensures that samples are independent and evenly distributed. Every training step a random sample is taken from the replay buffer, which is used as the training batch. As a result, DDPG is an off-policy algorithm, as the agent at the time of recording the experience was different from the agent at the time of training. The full algorithm for DDPG as used in this study can be found in Algorithm 1.

Algorithm 1 The DDPG Algorithm

Initialize critic and actor networks Q and μ with weights θ^Q and θ^μ
Initialize critic-target and actor-target networks Q' and μ' with weights $\theta^{Q'}$ and $\theta^{mu'}$
Initialize empty replay buffer R
for episode = 0, episode < M **do**
  Retrieve environment state s_0
  for t = 0, t < T **do**
    Generate action $a_t = \mu(s_t|\theta^\mu)$
    Perform a_t in environment and receive reward r_t and state s_{t+1}
    Store transition s_t, a_t, r_t, s_{t+1} in R
    Sample random batch of N transitions from R
    Calculate target values $y_i = r(s_{i,t}, a_{i,t}) + \gamma Q(s_{i,t+1}, \mu(s_{i,t+1})|\theta^Q)$
    Update critic by minimizing loss $L = \frac{1}{N}\sum_{i=0}^{N}(y_i - Q(s_i, a_i|\theta^Q))^2$
    Update actor with policy gradient $\nabla_{\theta^\mu} J = \mathbb{E}\left[\nabla_a Q(s, a|\theta^Q)\nabla_{\theta^\mu}\mu(s|\theta^\mu)\right]$
    Update the critic target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    Update the actor target network: $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for

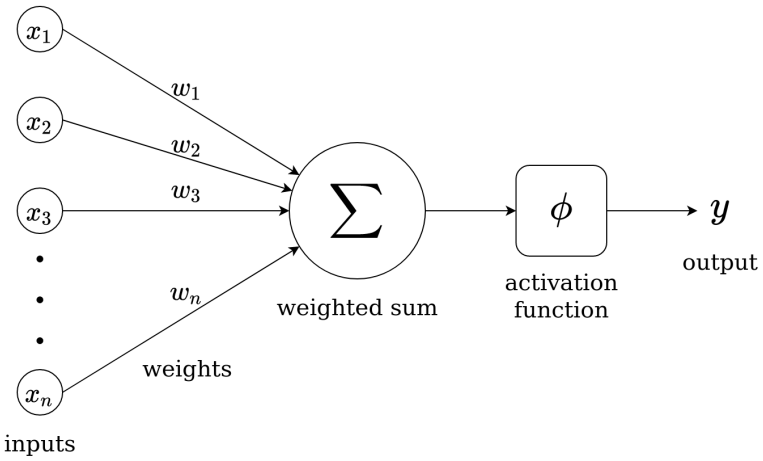


Figure 1: Conceptual model of a neuron

3.2 Neural Networks

A neural network consists of neurons that can receive inputs from and send trained outputs to other neurons. A neuron has a weight assigned to every input, which can be adjusted during training. All the inputs together add up to a 'weighted sum of inputs', which is used to determine the output y of the neuron:

$$y = \phi\left(\sum_{i=0}^{n} w_i x_i\right) \qquad \text{(Eq. 9)}$$

Here, n is the amount of inputs, w_i are the input weights and x_i are the input values. ϕ is the activation function that determines the output value, given the weighted sum. A model of a neuron is depicted in Figure 1.

There are several types of activation functions that each map the sum to the output value differently. The activation functions used in this project are ReLU and tanh.

- ReLU (Rectified Linear Unit) is linear if the sum x is zero or negative, the value of ReLU is 0. For the purpose of this project, ReLU is defined as $\max(0, x)$.
- tanh is the hyperbolic tangent, defined as $\frac{e^x - e^{-x}}{e^x + e^{-x}}$, and has a range between -1 and 1.

Neurons can be arranged in different architectures, and many architectures have been proposed and studied. Since our agent will receive image data as input, we studied architectures that can extract and use features from this image to determine the action output. We focused on convolutional layers for extracting features, and fully-connected layers for the final output.

For feature extraction from images, a 2-dimensional convolutional layer is often used. In these convolutional layers a relatively small matrix, known as the kernel, visits every pixel of the input data. If the kernel size is larger than 1, it will also cover pixels surrounding the pixel being visited. Every entry in the kernel corresponds to a weight, and every pixel covered by the kernel is multiplied by this weight. The sum of these multiplications is then used as the value of the corresponding pixel in the output.

Several convolutional layers can be arranged to form a convolutional neural network. In convolutional neural networks such as AlexNet [21] and InceptionV3 [1] many convolutional layers are used to extract features ranging from low-level features such as edges, curves and shapes to more abstract features such as text, cars or animals.

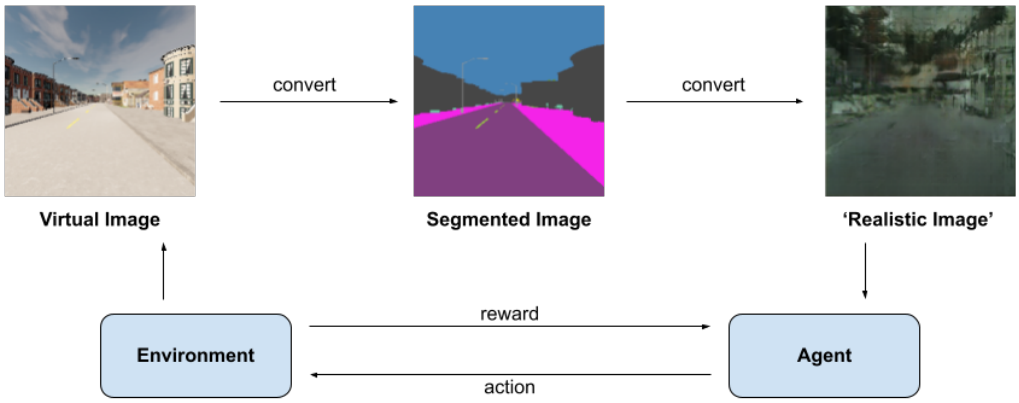


Figure 2: The image translation pipeline, with translations performed by Pix2Pix [16]

4 Method

As mentioned in the introduction, the goal of our framework is to reduce the simulation-to-reality gap for agents trained in a simulation. We aim to address this challenge this by passing the frames rendered by the simulator to an image translation model, which will translate the rendered frames to frames looking more similar to the real world. The agent will then use these translated frames as state input. The actions the agent decides to take will then be applied to the simulator and a reward will be returned to the agent. An overview of this process with Pix2Pix [16] can be found in Figure 2

4.1 Training the agent

The agent will be trained in a simulator using the Deep Deterministic Policy Gradient (DDPG) Algorithm described in Section 3.1. Currently, our agent only has one goal: to keep the car in the lane for as long as possible. To do this, it only needs to control the steering angle; throttle is regulated by the simulation. To encourage generalization, the agent will be roaming around in the five different maps that the CARLA simulator offers by default, rather than one. Training takes places over several episodes, which themselves are divided into many time-steps. At the beginning of every episode the episode reward is reset, a random map from the available maps is chosen, and the agent is spawned into this map.

The reward function in Equation Eq. 10 was designed to judge the agent's performance on every time-step. This function takes into account the angular difference between the vehicle and lane, and the vehicle's distance to the center of the lane. States with a smaller angular difference and distance are more desirable, and therefore receive higher rewards. For example, if in a time-step the vehicle is right in the center of the lane following the direction perfectly, the agent would receive a reward of 2 in that time step. If the agent is a meter from the center moving perpendicular to the lane direction, the agent would receive a reward of -1. The agent's experiences, consisting of the previous state, action taken, next state and reward received, are

stored in a replay buffer.

As input, the agent receives a frame buffer containing the last frame or frames. This input passes through the actor network described in Section 4.2, producing the steering angle as output. During training, the agent is free to roam the simulation environment with few restrictions. However, if the agent is involved in a collision or leaves its lane altogether, it receives a big punishment in the form of a negative reward and the episode ends. There is also a set amount of time-steps after which an episode always finishes, regardless of the agent's actions. All the rewards received during the episode together add up to the episode reward, which the DDPG algorithm attempts to maximize.

$$r_t = \begin{cases} 1 + cos(\phi) - |sin(\phi)| - |d_{center}| & \text{if vehicle in lane} \\ -100 & \text{if vehicle left lane} \end{cases} \quad \text{(Eq. 10)}$$

The agent is also trained every time-step. A random sample of the agent's previous experiences, stored in the aforementioned replay buffer, is taken as training data. The networks are then updated as described in the DDPG algorithm given in Algorithm 1.

4.2 Network Architecture

DDPG uses two separate networks: the actor network, responsible for predicting actions, and the critic network, which evaluates the decisions of the actor network. Actions are predicted using frames as input, each of which has been processed by the image translation pipeline. In this project we use two variations of actor- and critic networks that use different networks for feature extractions.

The first variation of the actor network is presented in Figure 4a. They take four frames as input, which are processed by a three-layer convolutional network as shown in Figure 3, each with 32 filters and a 3x3 kernel size. The combined output of the convolutional networks is batch-normalized to improve network stability, and is then connected to a fully-connected layer. This is where the decision-making will happen. The final neuron outputs the predicted steering angle. All convolutional layers and the first two fully-connected layers are followed by rectified linear (ReLU) activation, and the remaining fully-connected layers are followed by tanh activation. The intent is for the convolutional layers to extract features from the input images that are useful for determining the best course of action. Recognizing the road markings, for example, could be very beneficial.

The first variation of the critic network can be found in Figure 5a. One branch of this network is identical in architecture to the aforementioned actor network. However, it also has a branch that processes the action associated with the state input. The two branches are concatenated into a fully-connected layer, batch-normalized and finally connected to two more fully-connected layers, the last of which outputs the predicted Q-value of the given state-action input. All the layers that are not part of the actor network branch are followed by ReLU activations.

The difference between the two variations of actor and critic networks is the approach to feature extraction. While the first variation uses three convolutional layers that are not pre-trained, the second variation uses the InceptionV3 [1] model for feature extraction. InceptionV3 is also a convolutional neural network, but much deeper than the three layers in Figure 3. This model is provided by Keras [22] and comes pre-trained on the ImageNet [23] dataset, which makes it suitable for general image recognition tasks. We include this model variation because we anticipate it to yield shorter training times and better results compared to the architectures in Figures 4a and 5a, as the feature extraction layers of the model do not have to be trained

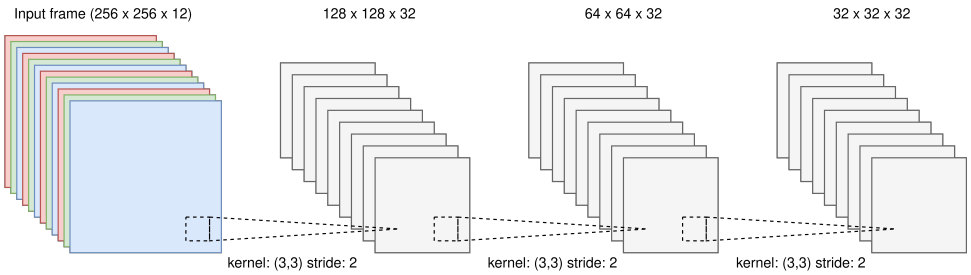


Figure 3: The convolutional network that processes every frame

from scratch. As the network is trained on single images, we only provide it with a single frame as input.

The critic network is similar in architecture to the actor network, but it contains some extra components to process the action input along the image input. Along with the convolutional layers it contains an extra fully-connected layer, which is then concatenated with the output of the convolutional layer into another fully-connected layer, which in turn is connected to the output neuron. The complete critic network can be found in Figure 5.

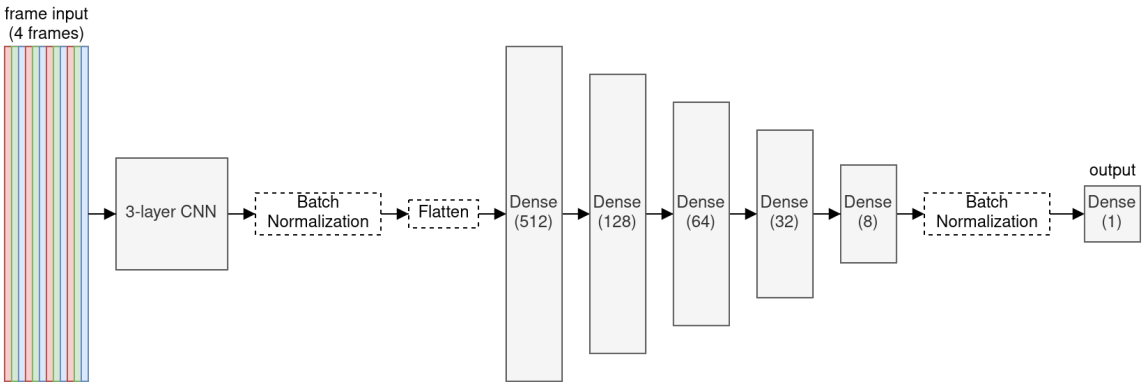
4.3 Image Translation

. To aid in addressing the simulation-to-reality gap, the agent does not receive the raw simulator images. First, the simulator output is processed by a set of image models. These models have previously been trained on image pairs of the source and target appearance, and their goal is to map the visual appearance of the simulator environment to the visual appearance of the environment in which the agent will be deployed.

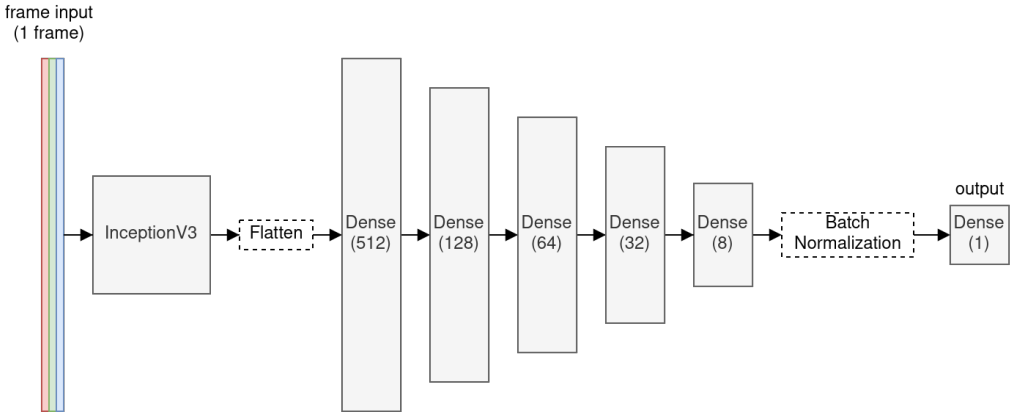
This mapping can happen in one or more translation steps, depending on what datasets are available to train the image model on. For example, if a dataset containing image pairs from the simulator and realistic environment were available, only a single translation model would need to be trained. In our case such a dataset does not exist, and as such an extra step is needed that translates the frames into an intermediary representation. In the context of autonomous driving, datasets with pairs of segmented scene data and real images are available, such as the Cityscapes [17] dataset. This segmented scene representation would be a suitable intermediary representation. We trained our first model on virtual and segmented frames we collected from the CARLA simulator, and trained our second model on the CityScapes [17] dataset. An overview of the pipeline can be seen in Figure 2.

It is worth mentioning that the CARLA simulator does have a segmented image camera, which would allow us to skip one of the translation steps. We decided not to use this feature and opt for the 2-step translation pipeline because the goal of this project is to propose a framework for training agents in any simulation, not just CARLA.

For the image translations we will be using the Pix2Pix framework designed by Isola et al. [16]. Pix2Pix uses a conditional generative adversarial network (cGAN) architecture, which consists of a generator and a discriminator network. These two networks are trained in an adversarial fashion. The generator is trained to generate images that are hard to distinguish from real images. Meanwhile, the discriminator is trained to determine whether an image is a real example or synthetic, i.e. produced by the generator. The architectures used for the generator and discriminator in this project are derived from [16] and can be found in Figures 6 and 7, respectively. The network of the generator is a so-called U-net [24] architecture: an encoder-decoder style network augmented with skip connections between encoder and decoder

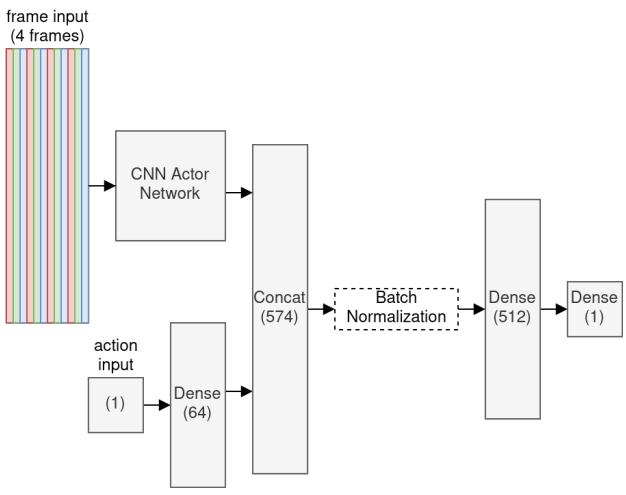


(a) The actor model with feature extraction through conventional convolutional layers. The '3-layer CNN' refers to the convolutional network shown in Figure 3.

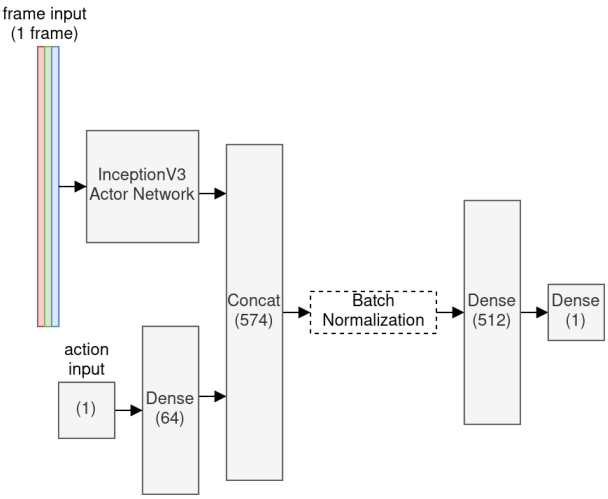


(b) The actor model with feature extraction through pre-trained InceptionV3 [1] network.

Figure 4: The variations of the actor networks, which are used to predict actions on a given state



(a) The critic model with feature extraction through conventional convolutional layers. The 'CNN Actor Network' block refers to the architecture in Figure 4a.



(b) The critic model with feature extraction through pre-trained InceptionV3 [1] network. The 'Inception Actor Model' block refers to the architecture in Figure 4b.

Figure 5: The variations of the critic network architecture used, which are trained to determine the 'goodness' of actions predicted by the actor networks.

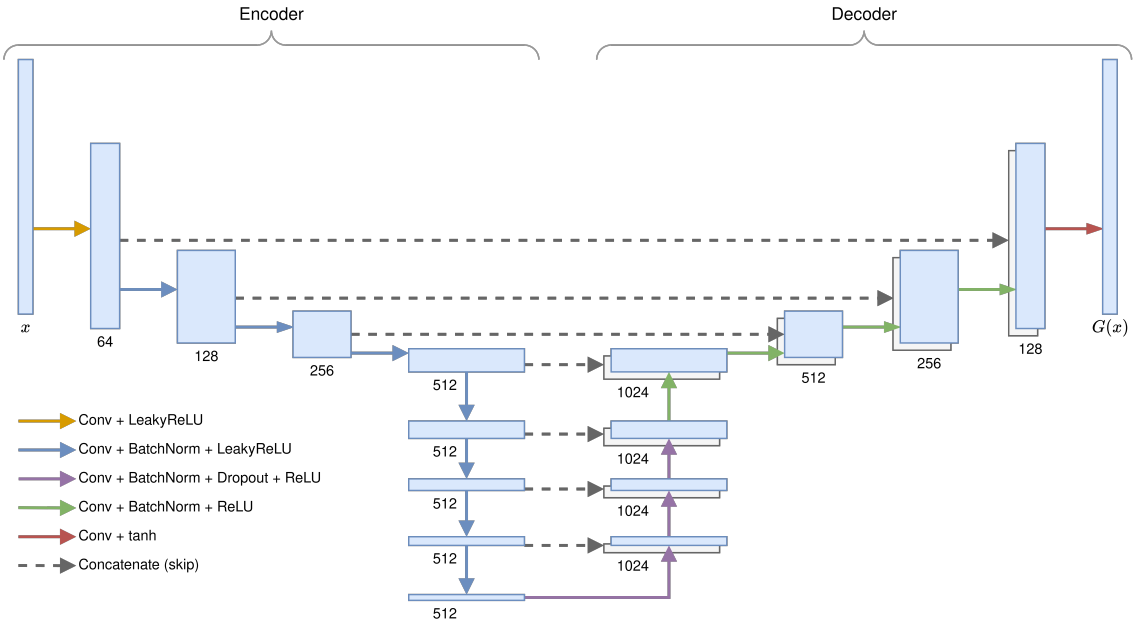


Figure 6: The network architecture of the Pix2Pix Generator, a so-called U-net [24], from [16]. The numbers describe the amount of feature filters.

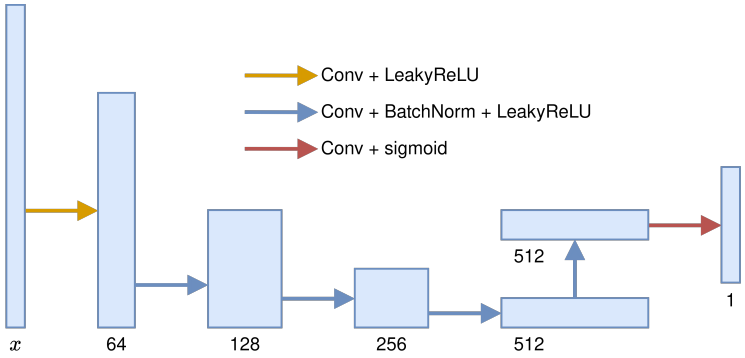


Figure 7: The network architecture of the Pix2Pix Discriminator, from [16]. The numbers describe the amount of feature filters.

layers. Using a U-net with skip connections was found to give "much higher-quality results" [16] compared to an encoder-decoder architecture without skip connections. The network of the discriminator was designed by the authors to penalize image structure only at an image patch level, meaning that if you look at the images as a whole, the generated images should resemble real images.

5 Experimental setup

The main issue that this sim-to-real framework attempts to address is the simulation-to-reality gap that that emerges when training an agent in a simulator. The works of Pan et al. [5] also focuses on this issue. Both of these projects train the agent with the Asynchronous Actor-Critic (A3C) algorithm, and use TORCS [3] as a simulation environment. This project uses the DDPG [6] algorithm to train the agent in the CARLA simulator. A major difference between A3C and DDPG is that the former is an on-policy algorithm, while the latter is an off-policy algorithm.

We think that the off-policy characteristic will lead to better exploration.

In Pan et al. [5], A3C outputs a probability distribution over a discrete action space. In our method, DDPG outputs continuous actions directly. We believe a continuous action space will lead to an agent that more closely emulates human steering behavior: after all, humans also steer in a continuous fashion to avoid jerky movements and loss of traction.

5.1 The CARLA simulator

The work of Pan et al. [5] uses TORCS [3] as a simulation environment, while we use CARLA. Compared to TORCS, CARLA is a more modern simulator that simulates an urban environment rather than a track environment, and therefore should match the features present in the real world better than TORCS. CARLA also comes preloaded with a few maps: the images collection and training agents roam the 'Town01', 'Town02', 'Town03', 'Town04' and 'Town05' maps.

5.2 Image Collection

For training the image models, a TensorFlow implementation by Christopher Hesse [25] of the Pix2Pix program developed by Isola et al. [16] is used. The goal is to make the image frames from the simulator look more like the real world, but unfortunately a dataset containing pairs of CARLA and real images does not exist. We therefore perform the translation in two steps:

1. Convert the virtual frames to a segmented scene representation
2. Convert the segmented scene representation into a 'realistic' frame

This means we need to train two models and therefore two datasets are required: one dataset with the simulator frames and corresponding segmented images, and one dataset with segmented images and corresponding 'realistic' images.

The first dataset is the most challenging, as we need to collect it ourselves. Fortunately, the CARLA simulator already has a sensor that can capture the segmented scene representation that we need. A script was created in which a vehicle drives around the maps that are provided in the CARLA simulator. The route the car takes is random, but the car does stay on the road. A set amount of virtual frames and corresponding segmented frames can then be recorded at regular intervals. After some processing, these frames can be used as input data to train the first image model. The dataset we use for the second model is the 'gtFine' package of CityScapes dataset [17].

10,000 image pairs were collected from CARLA for training the simulation-to-segmented image model. Both RGB and segmented image were recorded in a 256x256 resolution with 1-second intervals. The Cityscapes dataset used consists of 2975 image pairs; the images are also downscaled to a 256x256 resolution prior to training.

5.3 Training the image models

For the image translation models we use the Pix2Pix framework described in Section 4.3. The virtual-to-segmented model is trained on the collected CARLA frames for 50 epochs. The segmented-to-real model is trained for 100 epochs on the aforementioned CityScapes dataset.

5.4 Training the agent

The training of the self-driving agent follows the DDPG algorithm described in Section 3.1. We intend to train four different models with otherwise identical training parameters:

1. A model with feature extraction through the three-layer convolutional network, as shown in Figures 4a and 5a. Image translation will *not* be applied to the input frames. We refer to this model as **CNN-NoTranslation**.
2. A model with feature extraction through the three-layer convolutional network, as shown in Figures 4a and 5a. Image translation *will* be applied to the input frames. We refer to this model as **CNN-Translation**.
3. A model with feature extraction through the InceptionV3 network, as shown in Figures 4b and 5b. Image translation is *not* applied to the input frame. We refer to this model as **Inception-NoTranslation**.
4. A model with feature extraction through the InceptionV3 network, as shown in Figures 4b and 5b. Image translation *will* be applied to the input frame. We refer to this model as **Inception-Translation**.

The actor and critic networks as described in Section 4.2 are created and initialized, as are their respective target networks. A replay buffer that can contain 10,000 transitions is initialized, and the training loop starts. Each model is subjected to 500 training episodes. Episodes end when the vehicle departs its current lane.

Every training step the network is trained on a batch of 32 experiences with an actor learning rate of 0.0001 and a critic learning rate of 0.001 using the Adam optimizer. The target networks are updated with an update rate (tau) of 0.001. Finally, a discount factor of 0.99 is used to calculate the target values.

Every time-step the actor network predicts a steering angle based on the simulator's frame input, which has been translated by the image translation models. This action is applied to the simulation and the resulting state is fetched. The reward function found in Equation Eq. 10 determines the value of this state, taking into account the distance from the center of the current lane and the angle difference between the vehicle and the lane. The old state, action, new state and reward are then added to the replay buffer.

5.5 Validation

We can evaluate the effectiveness of each of the trained models by measuring its accuracy on an annotated real-life driving dataset. We use the annotated driving dataset from [26], which contains video frames of a driving trip performed by a human, annotated with the steering angle. This dataset is also used by Pan et al. [5], with which we can compare the accuracy. Comparing these accuracies allows us to verify how our combination of DDPG and CARLA performs compared to the combination of A3C and TORCS.

The real-life dataset contains steering angles in degrees, while the aforementioned work uses models that output discrete actions. The steering angles were therefore binned into three categories. Steering angles in range (-10, 10) were considered 'going straight', angles smaller than -10 'going left' and angles larger than 10 'going right'. Since our model outputs continuous data, we bin our model outputs in the same manner. Whether the bins of the prediction and annotation match up or not determines if a prediction is considered accurate. By comparing the prediction to the annotation, we can determine the accuracy of our self-driving models.

Our models are also compared to each other, which allows us to determine if there is a noticeable improvement in accuracy between the agents trained with and without image translation, and thus whether image translation is an effective method of bridging the simulation-to-reality gap. The comparison can also show if any of the feature extraction methods used may lead to better real-life accuracy.

6 Results

6.1 Image Translation Results

In Figure 8 we present a comparison between virtual images taken from the CARLA simulator, and the 'realistic' image that results when this virtual image is processed by the image translation models we trained. At first glance the translated images do seem to resemble the real world, but the realism falls apart upon closer inspection. Still, we believe that the translated images are more similar to what would be captured by a camera on the road than their virtual counterparts.

Interestingly, the translation models seem to have trouble painting in larger surfaces. This is especially apparent when a lot of sky is visible. We speculate the CityScapes [17] dataset used to train the segmented-to-realistic image model is the cause of this. The images contained in CityScapes tends to contain many smaller surfaces, and not too much empty road or sky. This theory is supported by the image pair captured in 'Town10HD' which does not contain large, empty surfaces and does not suffer from inpainting artifacts. In fact, this pair looks the most realistic of all the examples given in Figure 8, even though image pairs from this map were not present in the dataset used to train the image models.

6.2 Driving model training

Figure 9 shows the change in reward and episode steps over the course of their training. The reward reflects the quality of actions taken during the episode: better actions result in higher rewards. Better actions can also result in more rewards because the car stays in its lane longer, which means the episode is terminated later. The amount of steps per episode reflects how long the vehicle was able to stay in its lane. The models we trained behave very similar to one another, however no progress in either the reward per episode or the steps per episode is observed during our training. Unfortunately, due to time constraints, we were unable to train **CNN-NoTranslation**.

6.3 Comparing models

We compare the models we trained ourselves to the models trained by Pan et al. [5] by measuring the accuracy on Chen's annotated driving dataset [26]. They are compared to the following models trained by Pan et al. [5]:

1. Pan-Baseline, which was trained through reinforcement learning without image translation
2. Pan-SV, which was trained through supervised learning on Chen's dataset [26]
3. Pan-RL, which was trained through reinforcement learning with image translation

The results of the model evaluations are presented in Table 1. The **Inception-Translation** and **Inception-NoTranslation** obtained surprisingly high accuracies, considering the lack of training progress reported in 6.2. On top of that, the accuracies of these to models were identical.

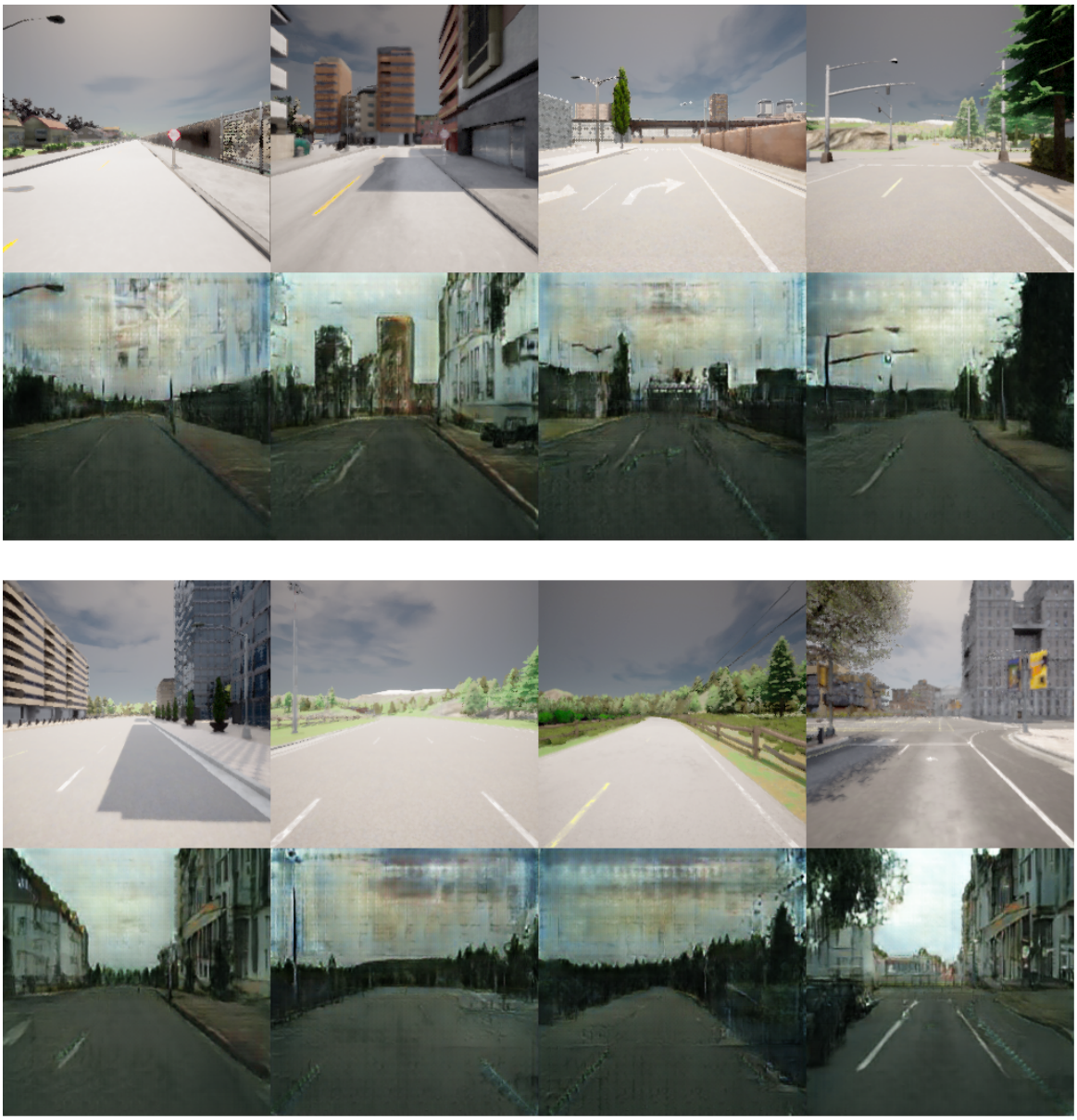
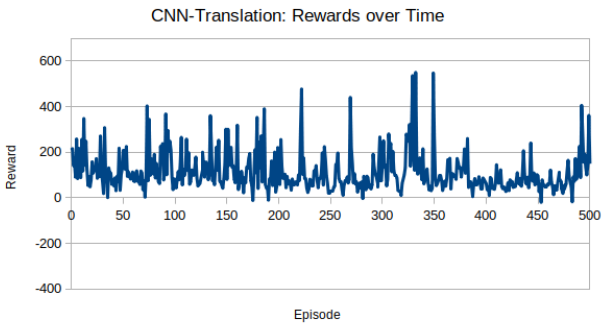
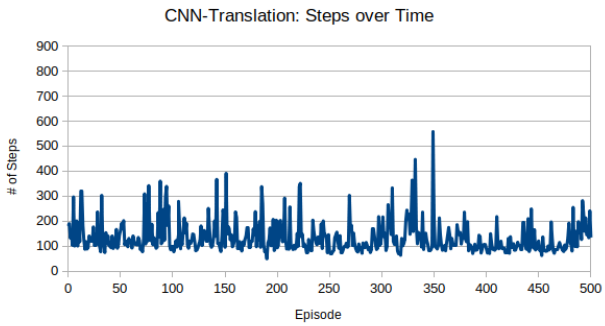


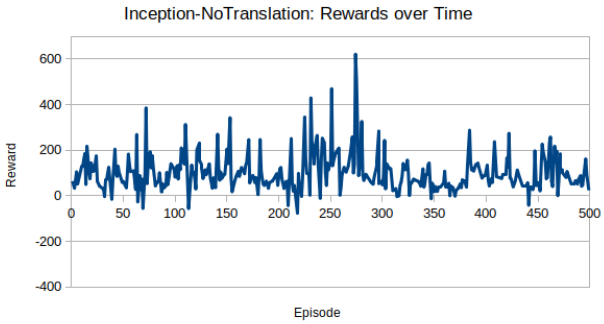
Figure 8: Comparison of virtual simulator output, captured from CARLA [18], and their translated counterparts, translated through the models trained in Section 5.3. Each pair is captured from one of the maps included with CARLA. From left to right, top to bottom: 'Town01', 'Town02', 'Town03', 'Town04', 'Town05', 'Town06', 'Town07', 'Town10HD'. Note that 'Town06', 'Town07' and 'Town10HD' were not included in the training dataset of the image translation models.



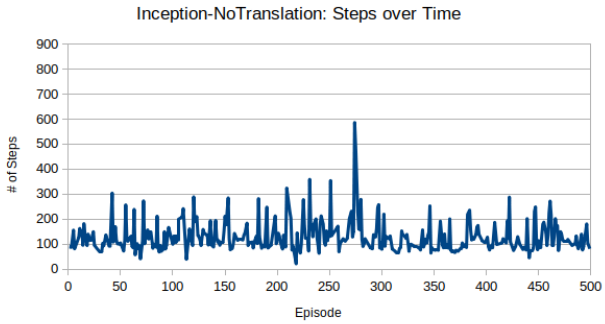
(a) Rewards per episode for the **CNN-Translation** model



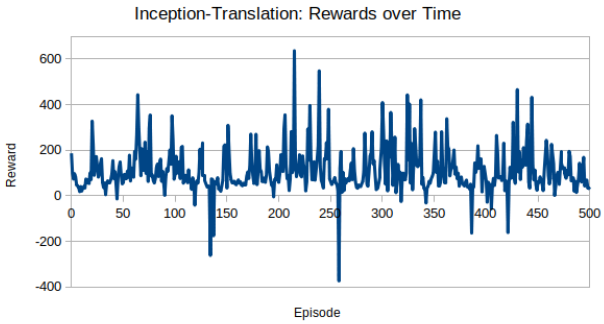
(b) Steps per episode of the **CNN-Translation** model



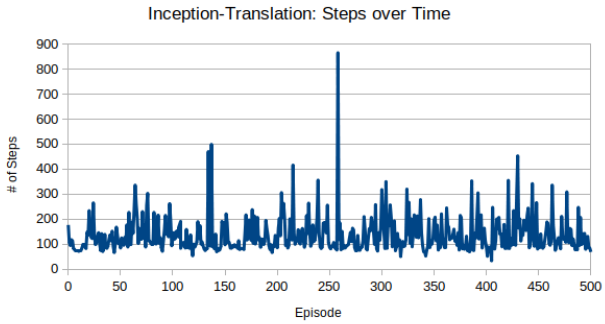
(c) Rewards per episode for the **Inception-NoTranslation** model



(d) Steps per episode of the **Inception-NoTranslation** model



(e) Rewards per episode for the **Inception-Translation** model



(f) Steps per episode of the **Inception-Translation** model

Figure 9: Change in Rewards and Steps per Episode over time for all our trained models.

Model:	Training method:	Accuracy on [26]:
Pan-Baseline	RL in TORCS	28.33%
Pan-RL	RL in TORCS with Image Translation	43.40%
Pan-SV	Supervised Learning on [26]	53.60%
CNN-NoTranslation	RL in CARLA	Unavailable
CNN-Translation	RL in CARLA with Image Translation	19.24%
Inception-NoTranslation	RL in CARLA	58.92%
Inception-Translation	RL in CARLA with Image Translation	58.92%

Table 1: Comparison of the accuracy of the models trained by Pan et al. [5] and ourselves on Chen's annotated real-life dataset [26]

Upon further investigation, this was due to these models always predicting a low steering value between 10 and -10 degrees, while a large portion of the annotated frames contain steering angles that are also in this range. We discuss these results further in Section 7.

7 Conclusions and Discussion

We present a framework that uses the CARLA simulator [18], Pix2Pix image translation framework, DDPG [6] algorithm and the neural network architectures from Section 4.2 in an attempt to train an end-to-end self-driving agent in a simulator while simultaneously addressing the simulation-to-reality gap. We compared our trained agents to the agents trained by Pan et al. [5]. While the image translation results look promising, with the scenes maintaining their overall structure while being closer in appearance to reality, the autonomous driving models trained by us did not seem to improve their behavior in the amount of steps we trained them, and did not improve on the models they were compared to.

Even though the models we trained showed little difference in reward during training, they did behave differently from each other during testing. Unlike the **CNN-Translation** model, the **Inception-NoTranslation** and **Inception-Translation** models predicted steering angles in a very small range. Because the binning method as described in Section 5.5 classified these values as 'going straight', and over half the frames in the used dataset [26] were also considered 'going straight', these models obtained a high accuracy rating. This does raise the question of how the models in [5] behaved during evaluation, but this information is unfortunately not included in the paper. It also indicates that this method of binning might not give the model with the best policy the highest accuracy score, as a policy of 'only going straight' would obtain a higher accuracy than all the driving models mentioned in Table 1.

The framework presented in this paper is not limited to training only the algorithm and networks presented in Sections 3.1 and 4.2. The framework is suitable for implementing, training and testing any neural network, and can thus be used for future studies. For example, it could be used to investigate whether the agents can be improved through more training episodes or a different network architecture; the actor- and critic networks in Figures 4b and 4b only use a single frame as input, and we considered using multiple frames as input instead, where each frame is processed by the InceptionV3 [1] separately. We also considered using a different classification network for feature extraction, such as InceptionV4 [27] or a ResNet [28].

We also believe that the image translation results could be improved further, by training the image translation model on a dataset that contains image examples with larger empty surfaces than CityScapes [17], or by combining multiple urban driving datasets into one training set.

References

[1] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1928–1937.

[3] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, *TORCS, the open racing car simulator*, `http://www.torcs.org`, 2000.

[4] E. Edmonds, *Aaa recommends common naming for adas technology*, https://newsroom.aaa.com/2019/01/common-naming-for-adas-technology/, Online; referenced on 17-08-2021, 2019.

[5] Z. W. Xinlei Pan Yurong You and C. Lu, "Virtual to real reinforcement learning for autonomous driving," in *Proceedings of the British Machine Vision Conference (BMVC)*, G. B. Tae-Kyun Kim Stefanos Zafeiriou and K. Mikolajczyk, Eds., BMVA Press, Sep. 2017, pp. 11.1–11.13, ISBN: 1-901725-60-X. DOI: 10.5244/C.31.11. [Online]. Available: https://dx.doi.org/10.5244/C.31.11.

[6] 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," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971.

[7] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 1, Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf.

[8] 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.

[9] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2174–2182.

[10] R. Zheng, C. Liu, and Q. Guo, "A decision-making method for autonomous vehicles based on simulation and reinforcement learning," in *2013 International Conference on Machine Learning and Cybernetics*, IEEE, vol. 1, 2013, pp. 362–369.

[11] Z. Huang, J. Zhang, R. Tian, and Y. Zhang, "End-to-end autonomous driving decision based on deep reinforcement learning," in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, IEEE, 2019, pp. 658–662.

[12] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2017, pp. 23–30.

[13] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall, "Learning to drive from simulation without real world labels," in *2019 International conference on robotics and automation (ICRA)*, IEEE, 2019, pp. 4818–4824.

[14] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 327–340.

[15] Y. Shih, S. Paris, F. Durand, and W. T. Freeman, "Data-driven hallucination of different times of day from a single outdoor photo," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, pp. 1–11, 2013.

[16] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[17] 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.

[18] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*, PMLR, 2017, pp. 1–16.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[20] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*, PMLR, 2014, pp. 387–395.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[22] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[23] 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*, Ieee, 2009, pp. 248–255.

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[25] C. Hesse, *Pix2pix-tensorflow*, https://github.com/affinelayer/pix2pix-tensorflow.

[26] S. Chen, *Autopilot-tensorflow*, https://github.com/SullyChen/Autopilot-TensorFlow.

[27] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.