

Curricular Internship Report

Resource Efficiency using Deep Q-Learning in Autonomous Driving

Eliandro Melo

Supervisor: Ricardo Cruz

Porto, 22 de Janeiro de 2024

Index

ABSTRACT	2
1. INTRODUCTION	3
1.1. Tools Used	3
1.2. Libraries Used	4
2. Related Work	5
3. Segmentation Model	5
3.1. Model	5
3.2. Dataset	6
3.3. Training	7
3.4. Testing	7
4. Deep Q Learning	8
4.1 Environment	8
4.2. Agent.....	10
4.3. Deep Neural Network.....	12
5. Discussion	14
5.1 Results	14
5.2. Future Work	15
6. Conclusion	15
7. Acknowledgements	16
8. References	17

Figures and Tables Index

Figure 1	5
Figure 2	9
Figure 3	10
Figure 4	11
Figure 5	13
Figure 6	14
Figure 7	14
Figure 8	16
Table 1	7

Abstract

This work introduces a resource-efficient approach to region selection for image segmentation in the context of autonomous driving, utilizing Deep Q-Learning, a reinforcement learning paradigm. This work seamlessly integrates a segmentation model, and an agent that iteratively selects regions for segmentation, using a Deep Q-Network. The environment represents an image divided into patches, with the agent dynamically choosing regions to enhance efficiency in the context of autonomous vehicle perception.

Deep Q-Learning is harnessed to empower the agent in learning an effective policy for region selection, aligning with the demands of resource-efficient autonomous driving systems. The proposed work, deeply rooted in Deep Q-Learning principles, serves as a robust foundation for further exploration and refinement in the specific domain of autonomous driving. This approach holds significant potential to advance resource-efficient region selection strategies in image segmentation tasks, contributing to the optimization of vision systems in autonomous vehicles through reinforcement learning.

Keywords

Deep Q-Learning, Deep Q-Network, Image Segmentation, Segmentation Model, Convolutional Neural Network, Region Selection, Agent, Environment.

1. Introduction

In the exciting world of self-driving cars, making decisions quickly and using computer power wisely is very important. One crucial task is figuring out what is in the images around the vehicle, and that's where image segmentation comes in. But sometimes, the usual ways of doing this can be slow and use up too much computer power.

In this project, we've come up with a new way of tackling this challenge. We're using something called Deep Q-Learning to make image segmentation in self-driving cars more efficient. Our approach works hand-in-hand with a segmentation model, a tool that helps the car understand what's in the pictures. We've found some ways of extracting useful information from the pictures, helping the agent get smarter and better at choosing the right parts to focus on.

We as humans, cannot process everything around us at once, we focus our attention selectively on parts of the visual space to acquire information when and where it is needed. The same happens with autonomous vehicles. That's when region selection comes in. The agent learns to pick the best areas in the images for segmentation. It's a bit like a game where the agent tries different strategies to get better and better at its job. We've made it clever by allowing it to explore new things sometimes and stick to what it knows works other times.

The main goal of the agent is to pick out the important bits in the images, in this case cars and people, while using as little computer power as possible. We've set up a system where the agent gets a little reward when it does well, and over time, it figures out the best ways to do its job efficiently.

This project is all about making self-driving cars not just smart but also good at managing their resources. We're excited about how this approach could make self-driving cars even more capable and help them make decisions faster and with less computer effort.

The implementation of the code is provided in a github repository [14].

1.1 Tools Used

In this project, we utilized a variety of tools to develop and implement our innovative approach for efficient image segmentation in self-driving cars. These tools played a crucial role in enhancing the capabilities of the agent, and optimizing resource management.

Programming language:

Python served as the programming language used for developing all the components of this project. Its versatility, extensive libraries, and community support make it the best suitable choice for implementing machine learning and deep learning algorithms.

PyTorch Framework:

PyTorch is an open source machine learning (ML) framework based on the Torch library, used for applications such as computer vision and natural language processing. Pytorch played a crucial

role in building, training and testing the segmentation model and Deep Q-Learning agent. It was chosen as the framework for this project because of its simplicity, flexibility, efficiency, but mainly because of its suitability for this project and its compatibility with the python language.

1.2 Libraries Used

In addition to the torch library, various libraries were used during the work, each with an important role in this project. These libraries were:

skimage: This library was used to read and resize the images from the dataset

torchvision: This library was used to retrieve the segmentation model used in this project.

torchmetrics: This library was used to measure the accuracy of the segmentation model.

albumentations: This library was used for the data augmentation used in the training and testing phases of the segmentation model.

tqdm: This library was used in the training and testing loop for the segmentation model.

time: This library was used in the training loop for the segmentation model.

os: This library was used to select a random image from the dataset for each of the agent's training loops.

collections: This library was used to create the agent's memory.

random: This library was used to help randomize some of the agent's actions.

2. Related Work

Reinforcement Learning:

Reinforcement Learning has been a widely explored area in machine learning, particularly in the context of decision-making and control systems. Classic Reinforcement Learning algorithms, such as Q-learning and SARSA, have been extensively studied in various applications (eg. [7], [8]).

Deep Q-Learning:

Deep Q-Learning (DQL) is a subset of deep reinforcement learning. Many notable works (eg. [1, 3]) have shown the ability of Deep Q-learning to handle challenging tasks. Extensions like Double D Q-Network (DDQN) (eg. [6]) and Prioritized Experience Replay (eg. [4]) have further refined the algorithm's stability and efficiency.

Additionally, various variants like Dueling DQN (eg. [5]) and Rainbow (eg. [2]) have been proposed to enhance the algorithm's performance by incorporating novel mechanisms and combining multiple improvements.

Researchers have explored the application of DQL in various domains, such as robotics, gaming, and autonomous systems. The adaptability and generality of DQL make it a versatile technique for solving complex decision-making problems.

Deep Q-Learning for image segmentation:

Deep Q-Learning for image segmentation has been relatively less explored than other tasks. Image segmentation is a challenging computer vision task that involves partitioning an image into meaningful segments or regions. As more researchers delve into this domain, we will see many breakthroughs and refinements in the application of DQL techniques to address the complexities of image segmentation (eg. [9, 10, 11]).

3. Segmentation Model

3.1. Model

In this project, the segmentation model serves as a fundamental component for the implementation of Deep Q-Learning in image segmentation. The chosen architecture for this model is DeepLabV3 with a ResNet50 backbone (Figure 1), a well-established model known for its effectiveness in segmentation tasks. The segmentation model is responsible for delineating regions of interest within an input image. These regions of interest are chosen by the reinforcement learning agent.

The model operates in binary mode, meaning that its focus is to accurately identify a single class of interest.

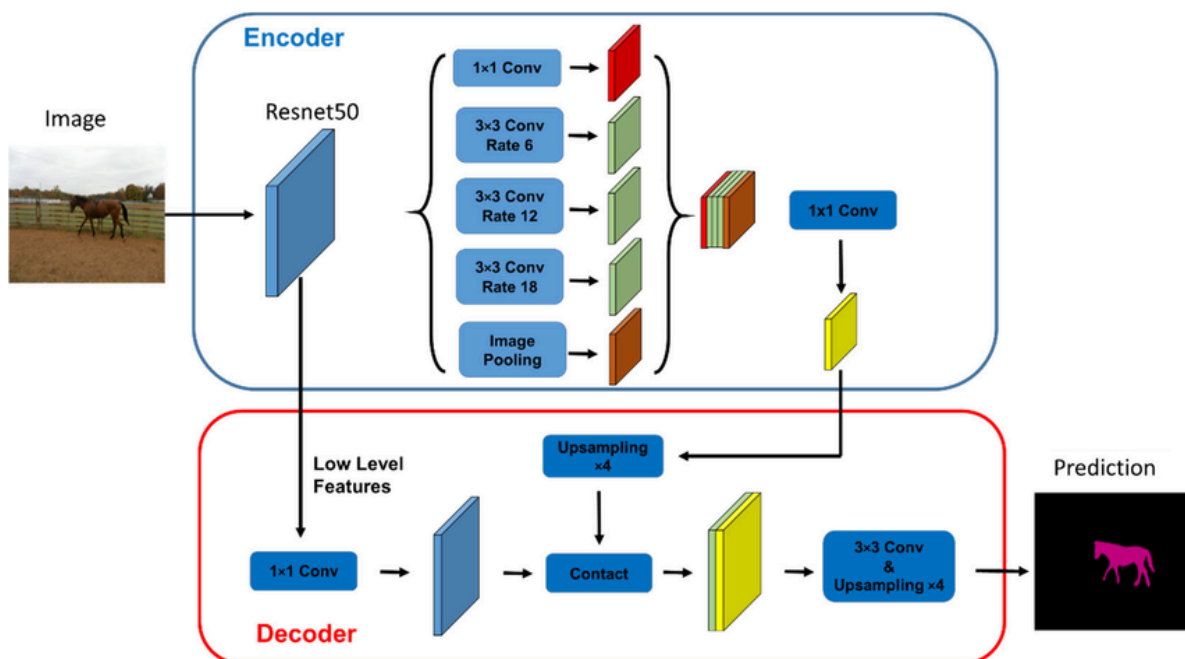


Figure 1: Segmentation Model

3.2. Dataset

It is crucial to choose an appropriate dataset for a project of this nature. The dataset used was the BDD10K_Binary a custom dataset derived from the Berkeley Deep Drive BDD100k dataset, the largest driving video dataset with 100K videos and 10 tasks to evaluate the exciting progress of image recognition algorithms on autonomous driving. [15]

The BDD10K_Binary is tailored specifically for binary segmentation tasks. It comprises a diverse range of images captured in various driving scenarios, providing a rich source of annotated data for the target segmentation class.

This dataset captures a diverse range of driving scenarios, including urban environments, highways, and varied weather conditions, ensuring that the model is exposed to a wide variety of situations. This diversity contributes to the model's robustness and generalization capability. Annotated data is crucial for training segmentation models, and BDD10K_Binary provides a rich source of annotated images. This richness allows the model to learn and generalize patterns associated with the presence of the chosen class in different driving contexts.

The chosen class for this project is a class composed by humans and vehicles. By focusing on this class, the segmentation model aims to accurately identify and delineate regions occupied by humans and vehicles in driving scenes.

3.3. Training

The training process is a crucial phase in developing the model. In this project, the training involves fine-tuning several crucial parameters to ensure the effectiveness of the training of the binary segmentation model. These parameters dictate the dynamics of the training process and have a significant impact on the model's performance.

The training script is configured to run for 1000 epochs, representing the number of times the entire training dataset is processed by the model. This extended training period allows the model to capture patterns, but continuous monitoring is necessary to prevent overfitting.

The chosen batch size is 32, indicating the number of samples processed in each iteration. While larger batch sizes can facilitate training, they also demand more memory.

The specified image size is 1024, influencing the resolution of the images presented to the model. Larger image sizes can capture finer details but may require increased computational resources.

The number of regions, set to 4 (which means 4x4 regions, that is, 16 regions), is a parameter intricately linked with the random cropping operation in the data augmentation pipeline. This parameter determines how the image is divided into distinct regions, influencing the granularity of augmentations applied during training. By choosing 16 regions, the augmentation process becomes more refined, contributing to a rich and varied set of training samples that helps the model generalize effectively across a broad spectrum of driving scenarios.

Before the training, the data passes for the data augmentation process using the albumentations library, a fast and highly customizable image augmentation Python library that can augment

images with diverse transformations. The goal is to enrich the training dataset, exposing the model to a wide range of variations in input data.

The optimization process and choice of loss function play important roles in training the segmentation model effectively.

The Binary Cross Entropy with Logits Loss is chosen as the loss function. This loss is well-suited for binary segmentation tasks, which involve classifying each pixel into one of two classes (foreground or background). BCEWithLogitsLoss combines the sigmoid activation function with binary cross-entropy, simplifying the training process for binary classification problems.

The chosen optimizer is the Adam optimizer, a popular choice for training neural networks.

3.4. Testing

The initial step involves preparing the test dataset by loading it using the specified parameters. The batch size, image size and number of regions are the same as the ones used for the training.

The albumentations library is also employed here to apply essential transformations to ensure that the data aligns with the model's expectations. Subsequently, a DataLoader is configured to efficiently load batches of test data.

To handle large images efficiently, a custom function is defined, breaking down images into smaller, manageable patches. This ensures that the model efficiently processes image regions during testing

The testing loop iterates through the Dataloader, generating predictions and computing evaluation metrics. As a measure, the Binary Jaccard Index is employed to evaluate the performance of the segmentation model. The measure is employed to quantify the accuracy between predicted binary masks and original masks.

During the project, models with different parameters were tested to observe the impact of changes in the number of regions and image size.

We can see on the table the difference in accuracy between the models to understand the reasoning of the choice for the model's parameters. (Table 1)

Image Size	Number of Regions	Epochs	Accuracy(Binary Jaccard Index)	Balanced Accuracy
2048	8x8	1000	73.1%	89.1%
1024	4x4	1000	82.6%	92.9%
512	2x2	1000	82.3%	93.5%

Table 1: Test results of the DeepLabV3 Resnet50 model with different parameters

4. Deep Q Learning

The algorithm chosen for reinforcement learning was Deep Q-Learning, explained in the following sections. To implement this algorithm, we have used the Snake game implementation [13], as the starting point.

4.1. Environment

The environment serves as the context in which the Deep Q Learning (DQL) agent operates. In DQL, the agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards (Figure 2). The environment encapsulates the task or problem that the agent needs to solve.

For this specific environment, the agent's task is to choose regions within an image for segmentation.

Arguments:

This environment takes 2 arguments.

The first argument is the model that we are going to use for the segmentation of his choice.

The second argument (nregions) represents the number of regions the image is divided into. This argument helps determine how finely the image is segmented.

State:

A state represents the environment at a given time. In this case, the state is represented as a tensor with dimensions (1, 1024, nregions, nregions), where 1024 represents the latent space dimensionality. Each element in the state tensor corresponds to the latent space features of a specific region selected by the agent. This representation allows the agent to capture relevant information about the chosen regions for segmentation..

Reward Calculation:

The environment is responsible for the reward calculation. The reward is a crucial feedback mechanism that guides the learning of the agent. In this environment, the reward is calculated based on the accuracy of the agent's segmentation.

The method responsible for calculating the reward compares the agent's segmentation (predicted mask) with the original mask, and the reward is normalized by the number of iterations. This normalization encourages the agent to optimize its decisions with fewer iterations.

Reset:

At the beginning of each episode the reset method is called to initialize the environment. It resets variables, including image regions, masks, and the agent's internal state, preparing the environment for a new episode. An episode is a sequence of interactions between an agent and the environment from initial to final states.

Step method:

The step method is a crucial component of the environment that gets invoked when the agent takes an action. Its primary purpose is to update the environment's current state, calculate rewards, provide feedback to the agent based on the action he has chosen, and verify if the agent completed an episode.

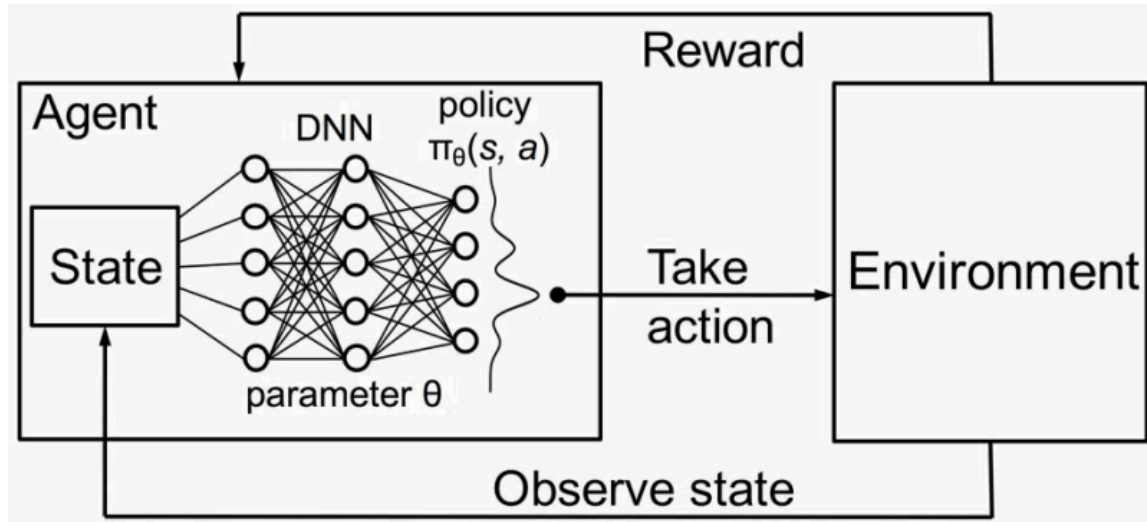


Figure 2: Example of the dynamic between the environment and the agent in Deep Q-Learning.

4.2. Agent

The agent is built around a Deep Q-Network (DQN), and its task is to choose the region that the segmentation model should focus on at a given moment. All this happens within the environment.

Training loop:

All of the agent's journey happens inside a loop.

Action selection process:

The agent can choose between 16 regions for segmentation. This decision is made using the DQN, determining the optimal policy for the agent and acting as the brain of the agent and tells him what action to choose. The goal of the agent is to find an optimal policy, one that leads to the best cumulative reward.

Exploration and Exploitation:

In the realm of Deep Q Learning, the agent faces a fundamental dilemma known as the exploration-exploitation trade-off. This challenge revolves around the decision-making process and dictates the balance between exploring new actions and exploiting known actions to maximize cumulative rewards.

Exploration:

Exploration involves trying out random actions to discover their effects on the environment. It allows the agent to gain a broader understanding of the task and potentially discover more rewarding actions. However, excessive exploration might lead to suboptimal decisions in the short term.

Exploitation:

Exploitation, on the other hand, involves choosing actions that the agent believes will return the highest immediate rewards based on its current knowledge. This strategy aims to capitalize on the information acquired during the learning process. While exploitation can lead to short-term gains, it may result in overlooking potentially superior actions that have not been explored.

To navigate this trade-off, the epsilon-greedy strategy was employed. During action selection, the agent chooses the action with the highest estimated Q-value most of the time (exploitation), but occasionally, with a small probability epsilon, it selects a random action (exploration). This approach ensures a balance between exploiting the agent's current knowledge and exploring new possibilities, facilitating a more robust and effective learning process.

In the training loop, as the agent obtains its Q-value estimates through interactions with the environment, the exploration-exploitation strategy adapts dynamically, using a formula (Figure 3).

Early in training, when the agent's knowledge is limited, more exploration is encouraged. As the agent becomes more experienced, the balance is shifted towards exploitation, allowing the agent to leverage its learned knowledge for optimal decision-making.

Achieving an effective exploration-exploitation balance is pivotal for the success of Deep Q Learning agents, enabling them to learn efficiently and make informed decisions in complex environments.

```
self.epsilon = max(20, 80 - self.n_observations)
```

Figure 3: Formula used to update the epsilon each iteration of an episode. The “self.n_observations” represents the number of iterations the agent has done.

Action:

After executing an action the agent calls the method ‘Step’, which we described earlier. In this method the agent receives crucial information for his training. Each one of these informations has an important:

Reward:

The goal of the agent is to maximize the reward (Figure 4), a crucial feedback mechanism that describes how he should behave. As described above, every interaction with the environment results in a specific reward.

```
def _calc_reward(self, mask):  
    return (mask == 1).float().mean().to(device) / self.iteration
```

Figure 4: Formula used to calculate the reward. The “self.iteration” represents the number of times the agent selected a region.

Score:

The score is used as a metric whose sole purpose is to assess the improvement of the agent through each episode. The agent doesn't use this metric during his training process.

It is calculated as the cumulative reward obtained at each iteration until an episode is done, and reflects the overall success of the agent and the segmentation model.

Done:

The done flag marks the end of an episode of the agent's training loop. It is crucial to set this flag cautiously. If the flag is set to true prematurely, the agent will end the episode earlier than expected, and this will affect his overall performance. On the other hand, if the flag is not set to true at the right moment, the agent may end up in a very long, or even infinite episode.

Replay memory:

After each action, experiences are collected and stored in the replay memory. Usually, each experience consists of tuples encompassing the state, action, reward, next state. In this work we added to each experience a flag denoting the completion of the loop.

Once the loop is done, a maximum of 1000 random experiences are selected from the memory and are used to train the agent. In this work, the size of the replay memory defined is 100 000 experiences.

The agent employs a replay memory mechanism to store and recall experiences. This replay memory is crucial for breaking the temporal correlation between consecutive experiences, enhancing the stability of the training process.

The replay memory is trained separately. As opposed to the normal agent's training, which happens every iteration, this training is only made once all of the iterations are done (the agent chose all of the regions available).

4.3. Deep Q Network

In this project the Deep Q Network (DQN) is a convolutional neural network (CNN) designed for the task of region selection within the Deep Q Learning framework.

The DQN used in this work (Figure 8), consists in three layers, each followed by a rectified linear unit (ReLU) activation function. This design allows the model to capture complex patterns and representations from the input data effectively.

The model receives as an input a tensor representing a state, and returns as an output the q values of all the actions possible, in this case 16 actions. Each value represents the expected cumulative reward for taking a particular action in a given state (the input). We can think of the Q-values as the quality of each potential action.

4.3.1 Training

Training the DQN involves two distinct phases: regular agent training and long training, responsible for training the replay memory.

Both training phases happen within a class named 'QTrainer.' The QTrainer class takes the DQN, a learning rate, and a gamma value . The discount rate, γ , influences how much the agent values future rewards in the learning process.

The chosen loss function for training is Mean Squared Error (MSE) Loss, a commonly used metric for this type of task.

The training process is done by two methods: 'train_step' and 'train_step_long.' The 'train_step' method takes the current state, the action chosen by the agent, the resulting state after the action, and a flag indicating if the episode has ended. On the other hand, 'train_step_long' receives experiences and is responsible for training the replay memory of the agent.

At the heart of the training process lies the Bellman equation (Figure 5), a pivotal component for updating the Deep Q-network during training. This equation plays a crucial role in adjusting the Q-value function, which represents the anticipated cumulative future rewards for undertaking a specific action in a given state. The iterative application of the Bellman equation refines the Q-network's understanding, enabling it to make more informed decisions over time.

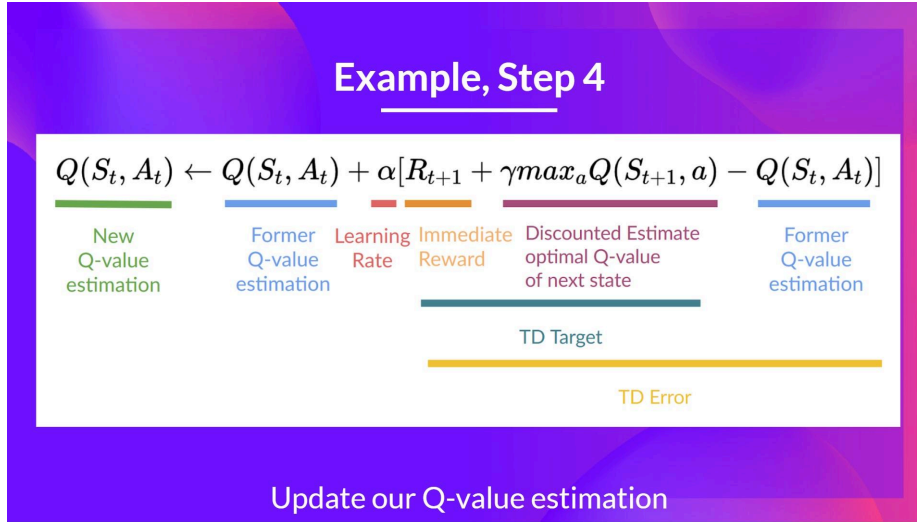


Figure 5: Bellman Equation [12]

5. Discussion

5.1. Results

In our experiments, the performance of the proposed Convolutional Neural Network (CNN) for image segmentation was subject to constant evaluation. We chose a DeeplabV3 model with a resnet50 backbone. This is a CNN architecture designed for semantic segmentation tasks, that is available in pytorch's torchvision library. While the results (Figure 7) obtained were insightful, it is important to note that our approach did not outperform the state-of-the-art models.

A major factor that led us to these results, was the choice of using the pre-built model over building a custom CNN model. This choice was made particularly due to the complexity of creating a custom model that would surpass the ones available in the torchvision library.

For the region selection task, a custom model was built (Figure 8), this model served as the Deep Q-Network (DQN). Despite the fact that the results (Figure 7) obtained with the model were satisfactory, previous works (eg. [11]) have shown how the Deep Recurrent Attention Model (DRAM), can be superior in tasks of this nature. Despite the acknowledged advantages of DRAM, the decision made was not to employ it in the current work. The primary reason for this choice was the complexity associated with DRAM. Implementing a Deep Recurrent Attention Model would pose much more challenges when compared to the chosen custom DQN architecture.

Despite not surpassing the state-of-the-art models, our results provide valuable insights into the complexities of combining Deep Q-Learning with region selection. The acknowledgement of the challenges posed by the DRAM and other architectural choices, lays the ground for future improvements that could face the challenges posed by the DRAM and other architectures.

Another key factor that led us to not surpassing the state-of-art results was the time constraint. A work of this nature demands a large period of time for improvement and completion.



Figure 6: Some results from the segmentation model's training

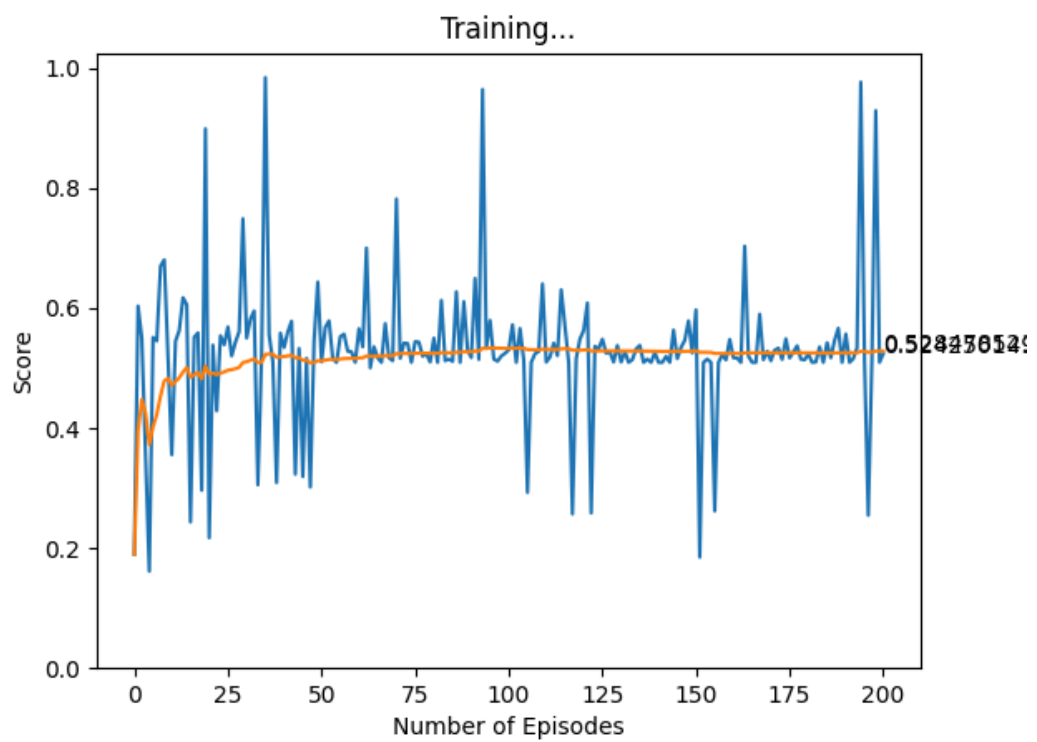


Figure 7: Results of the agent's training

5.2. Future Work

To address the limitations identified in our experiments, future works could explore the integration of a more efficient segmentation model. As noted above, for the segmentation task it was chosen a DeeplabV3 model with a resnet50 backbone. Instead of simply choosing a pre-built model available for us, opting for a well-built custom model using CNNs could lead us for much better segmentation results (eg. [9]). Another alternative would be using a DRAM. This model has proven itself, outperforming various CNNs (eg. [11]). Integrating such models within this work will contribute to a more accurate image segmentation.

Another path for improvement lies in fine-tuning the hyperparameters during the agent's training phase. In our experiments, we utilized a fixed set of hyperparameters, including the learning rate, discount rate, and epsilon decay rate. In future works we can have a more detailed exploration of each hyperparameter to identify optimal configurations for the specific task of region selection. Adjusting the learning rate could influence the convergence speed and stability of the training process. The discount rate, representing the agent's consideration of future rewards, could lead to a and the epsilon decay rate, controlling the exploration-exploitation trade-off, are critical factors in shaping the learning behavior. Fine-tuning these hyperparameters based on task-specific requirements could lead to significant improvements in the agent's performance.

For the Deep Q-Network, which represents the agent's brain, we can explore the possibilities of using a DRAM. The DRAM has demonstrated its efficacy by outperforming various Convolutional Neural Networks (CNNs) in relevant tasks (e.g., [11]). Considering the DRAM's ability to capture sequential dependencies and attend to specific regions of interest, its integration into the work would enhance the agent's decision-making process and, consequently, lead to more informed and accurate actions.

6. Conclusion

In this report, we presented a novel approach to enhance resource efficiency in image segmentation for autonomous driving using Deep Q-Learning. The integration of a Deep Q-Learning agent, responsible for iteratively selecting regions for segmentation, demonstrated the potential to optimize the computer vision system of autonomous vehicles through reinforcement learning.

In conclusion, this work contributes to the ongoing exploration of Deep Q-Learning (DQL) in region selection for autonomous driving. The proposed methodology, despite not outperforming existing models, provides a foundation for future advancements in resource-efficient region selection strategies. The combination of DQL and region selection holds promise for optimizing computer vision systems in self-driving cars, paving the way for more capable and efficient autonomous vehicles in the future.

7. Acknowledgements

I would like to thank my supervisor, Ricardo Cruz, for the immense help provided during the project. I would also like to thank InescTec for providing all the necessary resources for the completion of this work.

This work was supported by Portuguese National Funds, through AICEP, E. P. E. and the Innovation and Digital Transition Program (COMPETE2030). Project ATLAS - Trusted Autonomous Navigation, with Funding Reference 01/RPA/2022-C679908640-00009887.

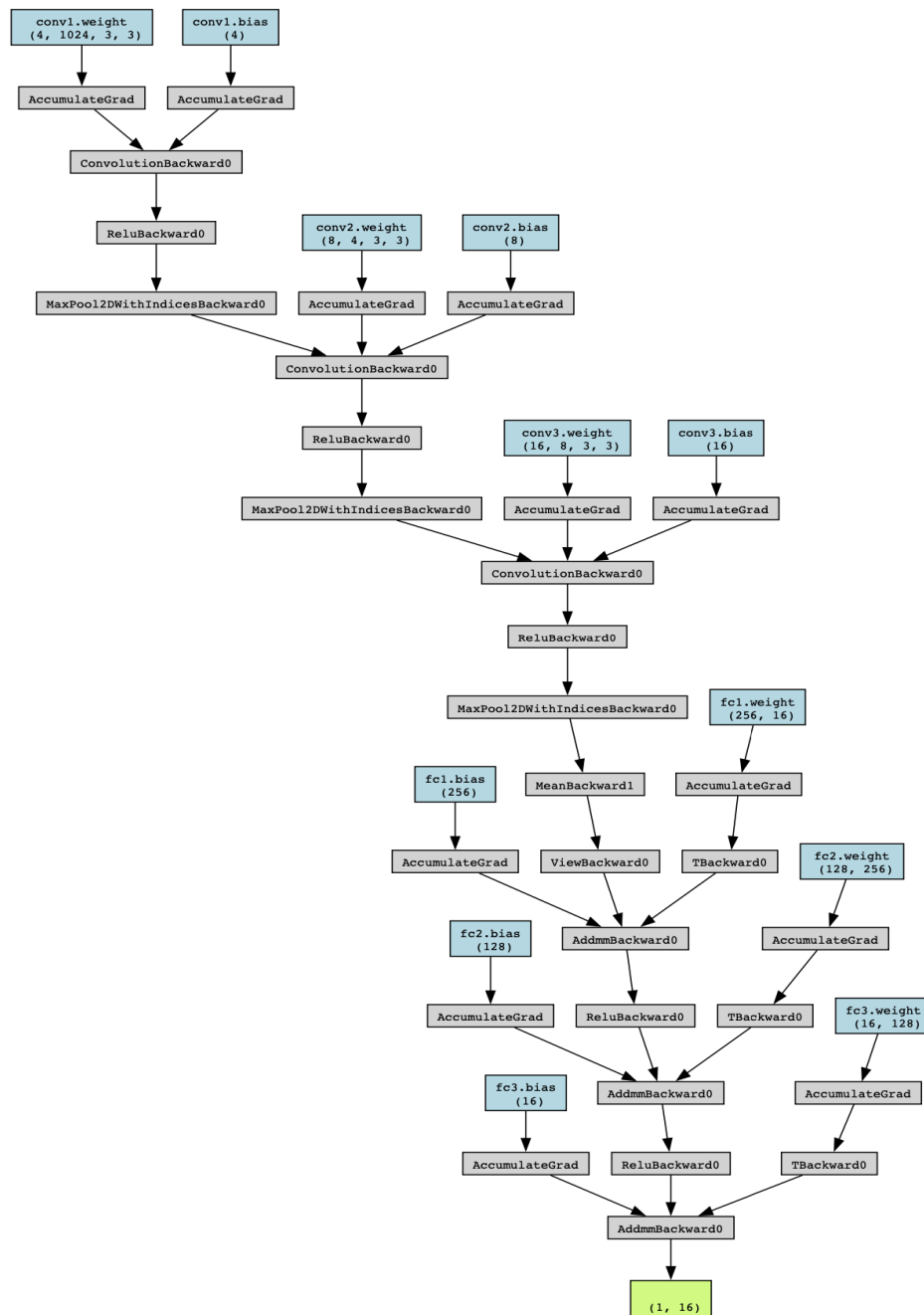



Figure 8: The Deep Neural Network used in this project

8. References

- [1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. In NIPS, 2012.
- [2] Matteo Hessel, Joseph Modayil, H. V. Hasselt, T. Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, M. G. Azar, David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. In AAAI Conference on Artificial Intelligence, 2017.
- [3] T. Lillicrap, Jonathan J. Hunt, A. Pritzel, N. Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. Continuous Control with Deep Reinforcement Learning. In International Conference on Learning Representations, 2015
- [4] T. Schaul, John Quan, Ioannis Antonoglou, David Silver. Prioritized Experience Replay. In International Conference on Learning Representations, 2015
- [5] Ziyun Wang, T. Schaul, Matteo Hessel, H. V. Hasselt, Marc Lanctot, Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In International Conference on Machine Learning, 2015.
- [6] H. V. Hasselt, A. Guez, David Silver. Deep Reinforcement Learning with Double Q-Learning. In AAAI Conference on Artificial Intelligence, 2015.
- [7] C. Watkins, P. Dayan. Q-learning. In Machine-mediated learning, 1992.
- [8] Taha Alfakih, Mohammad Mehedi Hassan, A. Gumaei, Claudio Savaglio, G. Fortino. Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA. In IEEE Access, 2020.
- [9] Arantxa Casanova, Pedro O. Pinheiro, Negar Rostamzadeh, Christopher J. Pal. Reinforced active learning for image segmentation. In International Conference on Learning Representations, 2020.
- [10] Usman Ahmad Usmani, J. Watada, J. Jaafar, I. Aziz, Arunava Roy. A Reinforced Active Learning Algorithm for Semantic Segmentation in Complex Imaging. In IEEE Access, 2021.
- [11] Mnih, V., Heess, N., & Graves, A. Recurrent models of visual attention. In Advances in neural information processing systems, 2014.
- [12] <https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm>
- [13]  Python + PyTorch + Pygame Reinforcement Learning – Train an AI to Play Snake
- [14] <https://github.com/ERJLM/Deep-Q-Learning-in-Autonomous-Driving>
- [15] <https://bdd-data.berkeley.edu/>