A SELF-DRIVING SYSTEM USING DOUBLE DEEP Q-LEARNING

by

Siraphop Prasertprasasna

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Mechatronics

Examination Committee:	Dr. Mongkol Ekpanyapong (Chairperson)
	Prof. Manukid Parnichkun
	Prof. Matthew N. Dailey
Nationality:	Thai
Previous Degree:	Bachelor of Engineering in Mechatronics
	Engineering Technology
	King Mongkut's University of Technology
	North Bangkok
	Thailand
Scholarship Donor:	His Majesty the King's Scholarships (Thailand)

Asian Institute of Technology
School of Engineering and Technology
Thailand
July 2021

AUTHOR'S DECLARATION

I, Siraphop Prasertprasasna, declare that the research work carried out for this thesis was in accordance with the regulations of the Asian Institute of Technology. The work presented in it are my own and has been generated by me as the result of my own original research, and if external sources were used, such sources have been cited. It is original and has not been submitted to any other institution to obtain another degree or qualification. This is a true copy of the thesis, including final revisions.

Date: 21 July 2021

Name: Siraphop Prasertprasasna

Signature:

ACKNOWLEDGMENTS

First and foremost, I would want to express my gratitude to my advisor, Dr. Mongkol Ekpanyapong, for his invaluable assistance, ideas, and support during the research process. It would be difficult to complete this thesis without his advice.

I would like to express my appreciation to the members of the examination committee, Prof. Manukid Parnichkun and Prof. Matthew N. Dailey, for their insightful remarks and knowledge sharing.

In addition, I would want to extend my thanks to Mr. Phawaphol Udompitayatorn, the AI Center's staff. He generously provided me with information and suggestions.

I would also want to show my gratitude to His Majesty the King's Scholarship of Thailand for funding my whole master's degree in AIT.

Finally, I would want to thank my parents for their unwavering support and aid during my education.

ABSTRACT

In the self-driving car industry, LIDARs and cameras have been used as sensors to drive a car. LIDARs can know the exact distance around a car, but when a car is driven by a human, a human does not know the exact distance like LIDARs. This thesis aims to create an AI that can drive like a human. Humans see an image when driving. Using only a camera was chosen to do this thesis, and the main AI algorithm is a double deep Q-network. When using a double deep Q-network, it requires a simulator to train the network. It will be hard if the network is trained in the real environment due to giving a reward. Training the network in a simulator will not be able to apply to a real car because the network has never seen a real environment before. A semantic segmentation was chosen to solve the problem. There are several semantic segmentation networks. PSPNet is the best choice to use in this case because it can provide quality segmented images, and it can run in real-time. In the end, a golf cart was used to drive in the left lane of a road. The golf cart has been controlled successfully.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figures **Page**

LIST OF ABBREVIATIONS

AI	= Artificial Intelligence
AIT	= Asian Institute of Technology
API	= Application Programming Interface
CNN	= Convolutional Neural Network
CPU	= Central Processing Unit
DQN	= Deep Q-Network
GPS	= Global Positioning System
GPU	= Graphic Processing Unit
HCN	= Hybrid Convolutional Network
IoU	= Intersection-over-Union
LIDAR	= LIght Detection And Ranging
mIoU	= mean Intersection-over-Union
OS	= Operation System
PSPNet	= Pyramid Scene Parsing Network
RAM	= Random Access Memory

CHAPTER 1
INTRODUCTION

1.1 Background of the Study

Over the past decades, automobile technology has been developed continuously. It began with the invention of the first steam-powered vehicle instead of horses, and the next generation of cars was powered by oil and gas. Nowadays, cars have changed to electric cars, and every car these days comes with many conventional systems, such as GPS and a rear-view camera, to assist the driver in various fields.

One of the conventional systems that the driver wants is a self-driving capability. The popular self-driving capability is a cruise control system that helps the driver without stepping on a pedal. In the present day, there is already a self-driving capability for automobile technology. The car will be driven and decided by AI.

1.2 Statement of the Problem

Since cars are the main vehicle that is used for traveling, the number of accidents has increased because of human errors. When an accident occurs, it causes death. Most accidents are caused by a driver's lack of focus on driving, such as sleepiness, drunkenness, and playing on a mobile phone. A self-driving capability will help a driver to prevent these problems.

Research papers about self-driving cars have been published increasingly, but there is no work that is 100 percent safe for today. News about accidents caused by self-driving vehicles has been seen periodically, and the self-driving systems of certain car brands, such as Tesla, are still not available in some regions of America and all regions of Thailand.

The hardware that is used to build the self-driving system is expensive. For example, LIDAR for visualization and distance measuring can be replaced by using cameras, and AI’s model optimization can replace expensive calculation hardware.

1.3 Objectives of the Study

This thesis aims to build a self-driving car that can move automatically from one location to another, as per the following details:

1. To create the best semantic segmentation model for road, lane marking, and object detection.
2. To create the best double deep q-learning model to control a car.

1.4 Scope and Limitations

Scope and limitations of this research as per the following details:

1. A car that is used in the real experiment is a golf cart.
2. A golf cart can be controlled in AIT.
3. A golf cart can be controlled on roads by having lane markings.
4. A golf cart can be controlled in the left lane.
5. A golf cart can be controlled in the daytime.
6. A camera installation in the front view of a golf cart is a vision for a self-driving system.
7. A driver must stay on the steering wheel for safety, and a golf cart can come back to control manually at any time.
8. The maximum speed of a golf cart is 8 kilometers per hour.
9. A golf cart can stay in its lane.
10. An AI's calculation device in real-time is a laptop.

CHAPTER 2
LITERATURE REVIEW

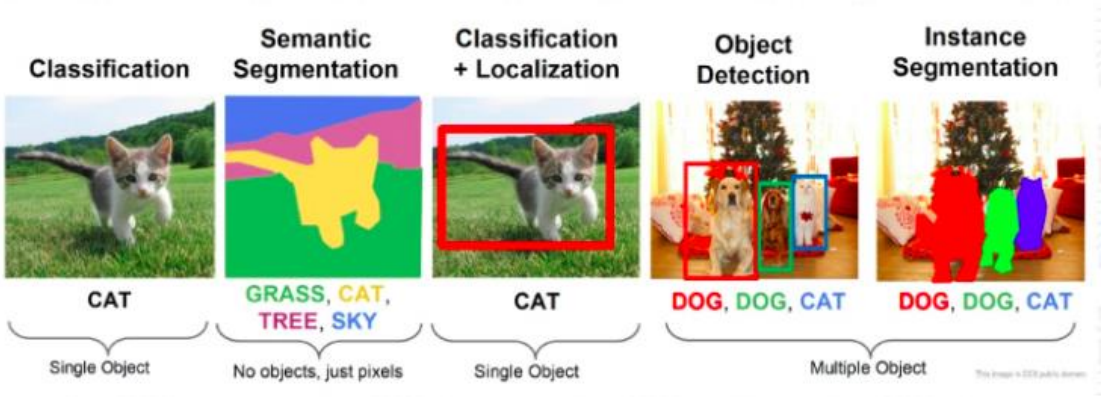
Chapter 2 describes algorithms that are used in this thesis and related work. The first part is a semantic segmentation to detect roads, lane markings, and objects. The next part is deep reinforcement learning, which is an algorithm for an AI to control a car, and the final part is simulators to simulate the road environment for training an AI.

2.1 Semantic Segmentation

Semantic segmentation is one of the image processing methods using the CNN architecture. There are 4 types of image processing methods using a CNN that consist of classification, object detection, image segmentation, and instance segmentation (see Figure 2.1).

Figure 2.1

Different Image Processing Methods Using CNN



Note. Classification using CNN can classify only one object, and CNN can predict this image as a cat, but CNN does not know the location of the cat in this image. Classification with localization is the same as object detection. The difference between them is the detection of a single object and multiple objects. Object detection can predict the location of the cat or dog in these images, and it can predict what those objects are, but it does not know the shape of the object that is detected. Semantic segmentation can predict every object for every pixel in that image, and it knows the shape of objects, but if there is more than one cat in that image, it will detect one cat. Instance segmentation can detect objects like semantic segmentation, but it can separate objects if there is more than one object for each class.

2.1.1 Related Work

Kai-Li Lim, Thomas Drage, and Thomas Bräunl presented a semantic segmentation to detect road and lane markings. They used SegNet, which is one of the CNN architectures for semantic segmentation. The reason why they use semantic segmentation is road and lane marking detection. They could not detect them by using only LIDAR. Errors were calculated as a percentage that corresponds to the area of each figure's road segments, as shown in Figure 2.2 and Figure 2.3.

Figure 2.2

Image Segmentation Result of a Clear Road Image Using SegNet

Input Output

Note. This result was predicted by SegNet, which was the lowest error of road pixels at 0.04 percent because the tested road's image was clear.

Figure 2.3

Image Segmentation Result of a Road Junction Image Using SegNet

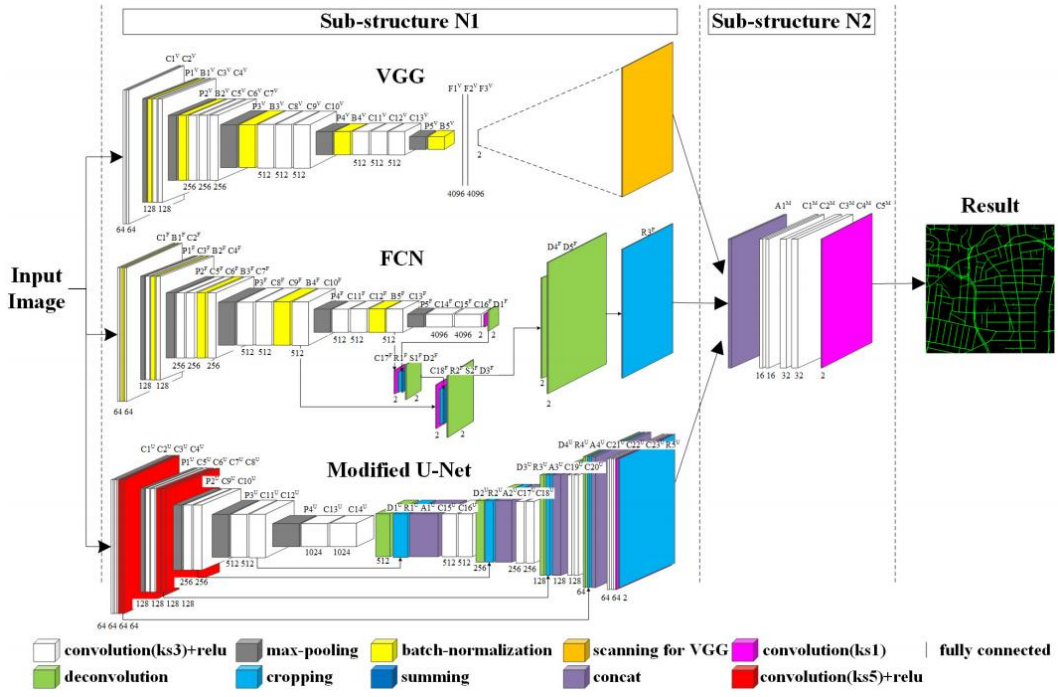
Input Output

Note. The SegNet predicted this outcome, which had the highest road pixel error of 16.16 percent because of poor road surface and illumination.

Ye Li, Lili Guo, Jun Rao, Lele Xu, and Shau Jin presented comparing popular architecture to segment roads from satellite images. The CNN architectures that they tested are VGG, U-Net, Modified U-Net, FCN, VGG+FCN, VGG+Modified U-Net, FCN+ Modified U-Net, SegNet, CasNet, and HCN. For modified nets, they adjusted the original architecture of that net by themselves to get the best accuracy. For HCN, they called another their architecture, which is the hybrid convolutional network (see Figure 2.4).

Figure 2.4

HCN Architecture for Road Segmentation



Note. The HCN architecture includes VGG, FCN, and Modified U-Net. The HCN can give the best mean pixel accuracy of 0.8355.

2.1.2 Semantic Segmentation Ranking

Robert Stojnic, Ross Taylor, Marcin Kardas, Viktor Kerkez, Ludovic Viaud, Elvis Saravia, and Guillem Cucurull created the Papers with Code website, which is a website to compare machine learning scores, and there is a semantic segmentation category. From Table 2.1, the most popular dataset for the contest is the Cityscapes test dataset. This thesis did some tests with 3 models, as shown in Table 2.2.

Table 2.1

Semantic Segmentation Dataset on Papers with Code Website

Dataset	Trend
Cityscapes Test Dataset	94
PASCAL VOC 2012 Test Dataset	57
PASCAL Context Dataset	43
ADE20K Validation Dataset	43
Cityscapes Validation Dataset	39

Note. The trend shows the number of models that have a benchmark score for each dataset. This table does not show all of the datasets on the Papers with Code website. It shows only the top 5 from 49 datasets.

Table 2.2

Cityscapes test Benchmark Score on Papers with Code Website

Rank	Model	Encoder	mIoU
1	HRNet-OCR	Hierarchical Multi-Scale Attention	0.851
36	DeepLabv3	ResNet-101	0.813
49	PSPNet	ResNet-101	0.784

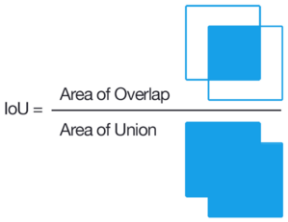
Note. This table shows only 3 models from 94 models because this thesis tested with these models.

2.1.3 Mean Intersection-over-Union

Mean intersection-over-union or mIoU is the most popular one for scoring semantic segmentation. For one class of prediction, it can be scored by IoU as shown in Figure 2.5. For multi-class prediction, it uses mIoU by averaging the IoU of each class.

Figure 2.5

Intersection-over-Union



Note. Intersection-over-union or IoU is calculated by an area of overlap divided by an area of union. These areas mean a ground truth area and a predicted segmentation area.

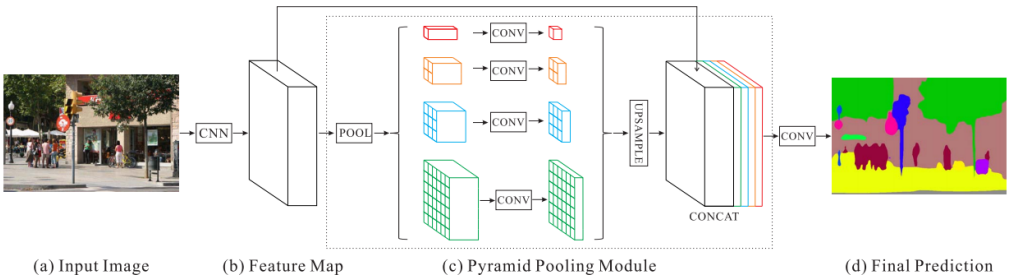
2.1.4 Pyramid Scene Parsing Network

Pyramid Scene Parsing Network or PSPNet is one of the semantic segmentation networks. PSPNet was invented by Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. In 2017, this network had a new record on the PASCAL VOC 2012 test dataset with mIoU 0.854, which is ranked 16th now. The overview of PSPNet is shown in Figure 2.6.

The full architecture of PSPNet can be found in Figure 2.9. The inside of PSPNet consists of ResNet-101, which is shown in Figure 2.8, and the inside of ResNet-101 consists of many Bottleneck blocks, which are shown in Figure 2.7. In this PSPNet, the input image size is 512x512 with a RGB channel, and there are 5 classes of objects.

Figure 2.6

Overview of PSPNet

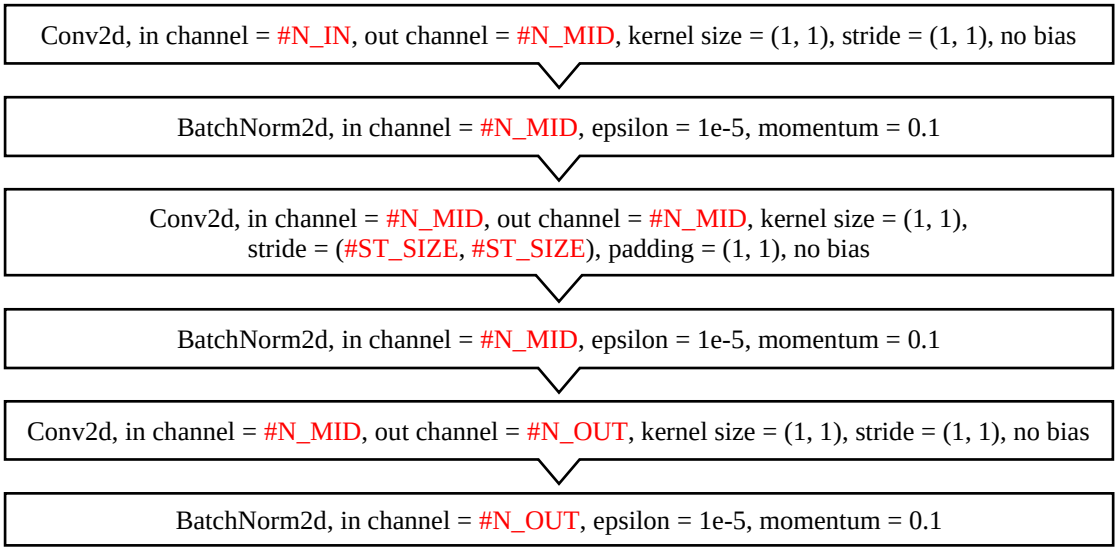


Note. The input image is fed into a feature map block. The feature map block is ResNet-101 or any encoder block. The output from the ResNet-101 is fed into the pyramid pooling module,

which is distributed to different sub-regions. At the last of the pyramid pooling module, it is upsampled, then concatenates the different sub-regions together. In the end, the output from the pyramid pooling module is fed into a convolution layer.

Figure 2.7

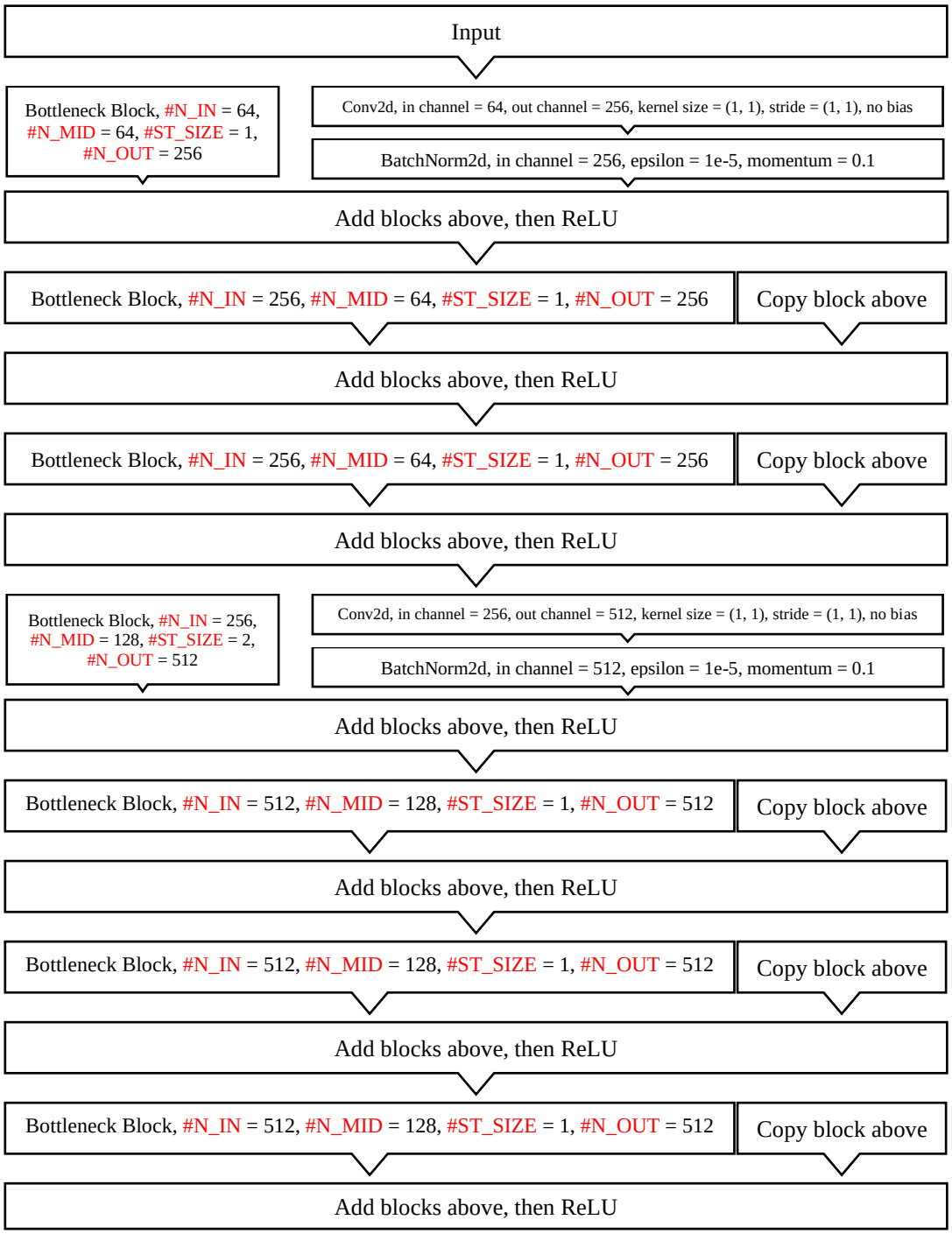
Bottleneck Block



Note. The bottleneck block contains 3 convolution layers which after each convolution layer, it is applied by a batch normalization layer. The bottleneck block is one part of the ResNet-101. There are 4 inputs of bottleneck block when used with ResNet-101. #N_IN is a number of the input channel, #N_MID is a number of the middle channel, #ST_SIZE is the size of the stride size, and #N_OUT is a number of the output channel.

Figure 2.8

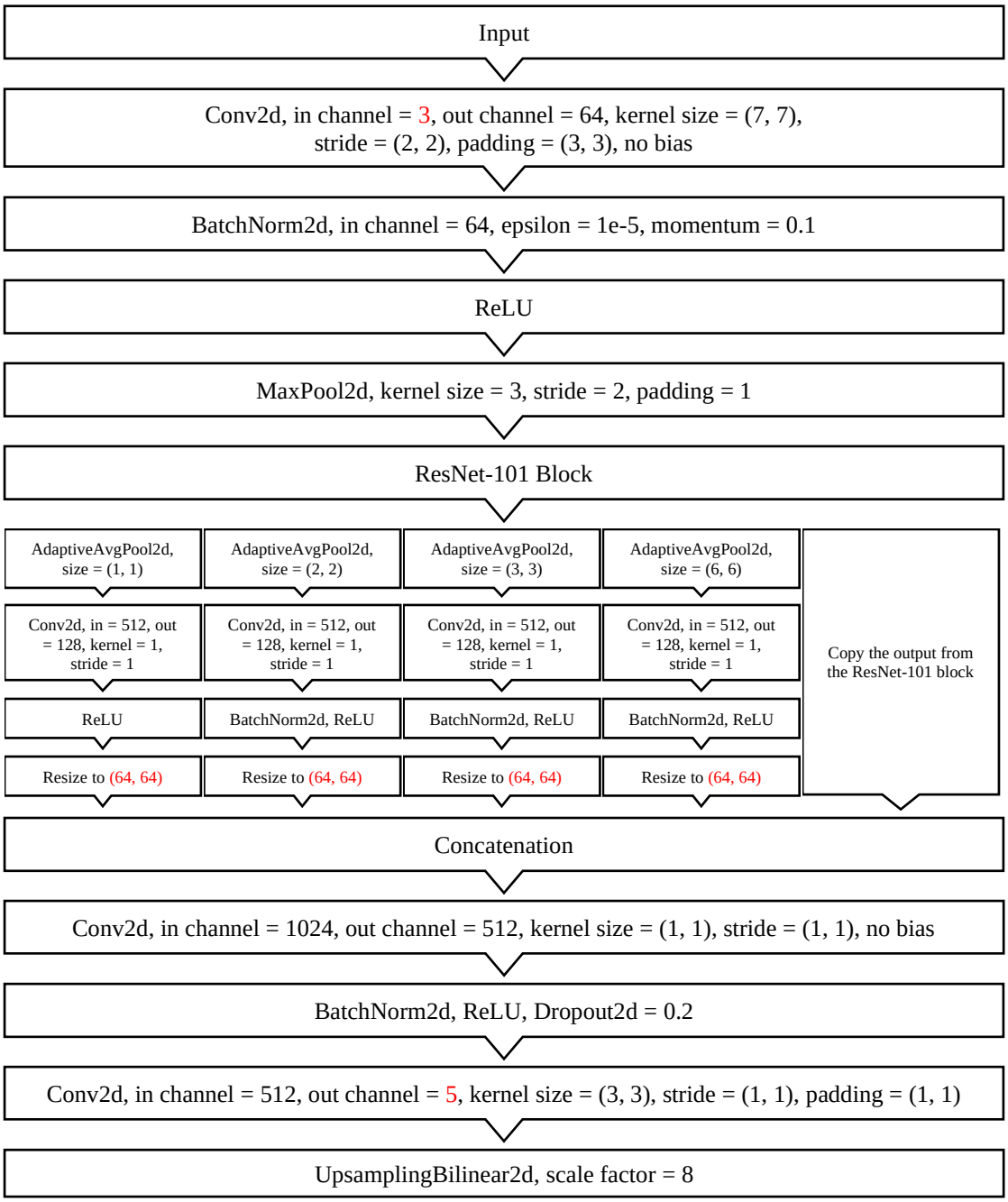
ResNet-101 Block



Note. There are bottleneck blocks inside ResNet-101. The bottleneck block is shown in Figure 2.7.

Figure 2.9

Full Architecture of PSPNet



Note. The red 3 number at the first convolution block means color channels. The numbers that can be changed depend on color channels and the image size of the input is marked in red. The AdaptiveAvgPool2d after the ResNet-101 block means resizing of the width and height channels of an image by averaging. The resize after that, the red (64, 64) is the original image input that is divided by 8. In this case, the original image input is (512, 512). The red 5 number at the last convolution block is the number of the object's class.

2.2 Reinforcement Learning

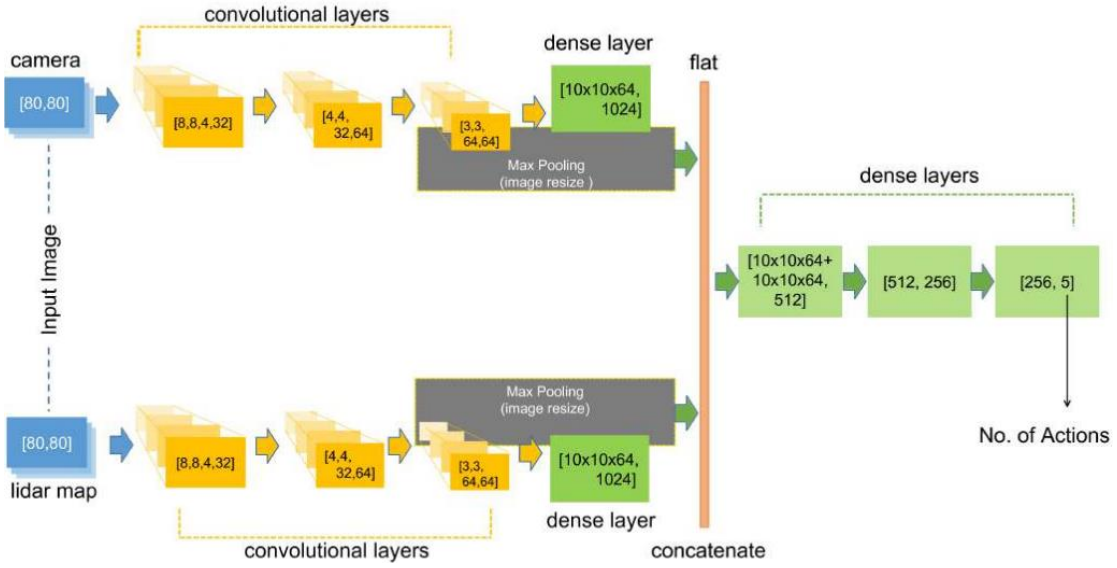
Reinforcement learning is one type of machine learning. There are 3 types of machine learning, which include supervised learning, unsupervised learning, and reinforcement learning. The first type is supervised learning. It trains a machine with known data and known responses. That machine is trained to predict new data that will give new responses. Examples of supervised learning applications are face recognition, weather forecasting, and optical character recognition. The second type is unsupervised learning. It trains a machine with known data and unknown responses to group the data. Examples of unsupervised learning applications are recommendation systems, buying habits, and grouping user logs. The last type is reinforcement learning. It learns from its mistakes to find the best solution. Examples of reinforcement learning applications are AI in video games, industrial simulation, and autonomous vehicles, and the most popular reinforcement learning is q-learning and deep q-learning.

2.2.1 Related Work

Abdur Razzaq Fayjie, Sabir Hossain, Doukhi Oualid and Deok Jin Lee presented a driverless car using deep q-learning. They have 2 inputs. The first input is 4 images from a front camera, which is an RGB image of size 80x80, and the second input is 4 data from LIDAR, which has a size of 80x80. Their overview is shown in Figure 2.10. They updated the network every 6000 steps. There are 5 actions for an agent to be trained, which are go straight, right, left, accelerate and brake. Keep going is doing nothing, and collect the reward at the current speed divided by 5. The Left collected a reward of -0.6. The right collected a reward of -0.2. Accelerate collected reward of +1. The brake collected a reward of -0.4, and if the car hit, the reward was -6. From these rewards, the network would learn how to accelerate and decelerate. They trained the network by simulation of an urban environment in the Unity Game Engine. Finally, they designed a car prototype based on their simulation experiments, which is capable of running a deep q-network in real-time.

Figure 2.10

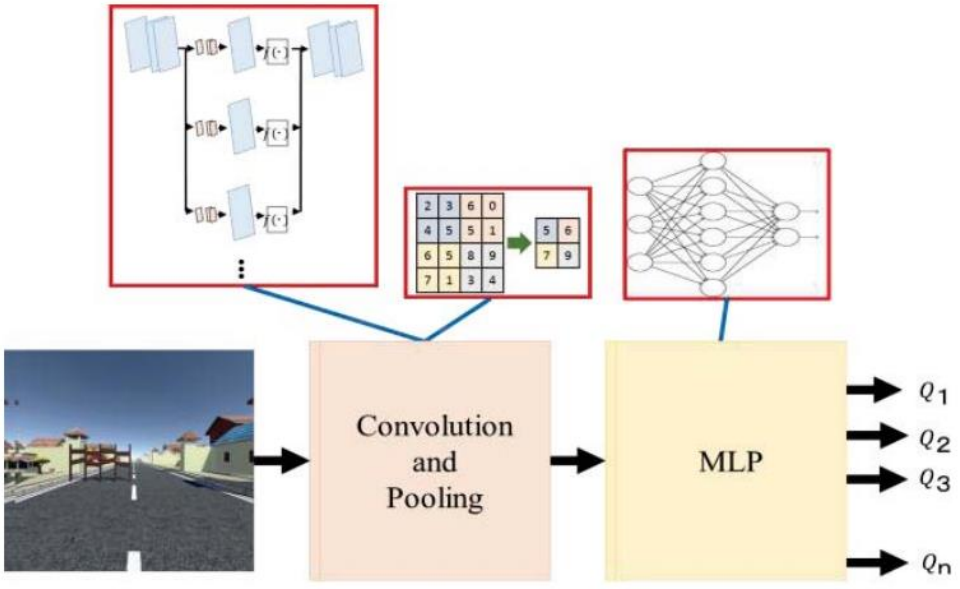
Deep Q-Network Architecture by Based on Fully Connected CNN



Note. Both inputs are fed into the neural network, and processed using the convolutional layers. After that, they concatenated them using flatten layers, preceded by a dense layer and a pooling layer.

Figure 2.11

CNN Architecture Gives Q Values



Note. An image was fed into CNN to get Q-values.

Takafumi Okuyama, Tad Gonsalves and Jaychand Upadhay presented an autonomous driving system based on deep q-learning. The input of the network is an RGB image as shown in Figure 2.11. They trained on a simulator which has a straight road bounded by footpaths on both sides, and obstacles are placed in random positions every 30 meters. There are 3 actions, which are steering 10 degrees to the left, keeping straight and steering 10 degrees to the right. They trained by repeating till the agent reaches the goal. In the state-action cycle, if the agent hits an obstacle or crosses one of the bounding lanes, the learning episode is discontinued, and the agent begins a new episode. The reward is the number of obstacles that it overcomes. They trained agents to avoid obstacles when driving at a constant speed of 10 meters per second, 15 meters per second and 20 meters per second. The highest distance of car speed at 10 meters per second is 10077 meters. The highest distance of car speed at 15 meters per second is 4907 meters, and the highest distance of car speed at 20 meters per second is 2173 meters.

Syed Owais Ali Chishti, Sana Riaz, Muhammad Bilal Zaib, and Mohammad Nauman presented self-driving cars using CNN and q-learning. The input of the CNN is an RGB image of size 32x24 that is resized from 320x240. There are 3 actions, which are forward, left and right. They tested it with signs on the road. There are 3 signs in this paper, which are the stop sign, the no-left sign and the traffic light for deep q-network. First, they used OpenCV cascade classifiers for sign detection. The results from OpenCV cascade classifiers would be the reward inspector, would not feed them into the deep q-network. For stop signs, if the stop sign was detected, and the car was moving forward, the reward value was -0.5, but if the car was stopped, the reward value was +2. If the stop sign was not detected, and the car was moving forward, the reward value was +0.05, but if the car was stopped, the reward value was -0.5. For a traffic light, if the red light was detected, and the car was moving forward, the reward value was -1, but if the car was stopped, the reward value was +0.01. If a green light or yellow light was detected, and the car was moving forward, the reward value was +0.01, but if the car was stopped, the reward value was -0.1. If no light was detected, and the car was moving forward, the reward value was +0.01, but if the car was stopped, the reward value was -0.1. Finally, they achieved 89 percent of training accuracy and 73 percent of testing accuracy from supervised learning.

2.2.2 Q-Learning

Q-learning is one of the reinforcement learning. It is an algorithm to optimize a Q-value by random action. The Q-value is changed for every learning step. An example of a Q-value is shown in Table 2.3 as a Q-table. The Q-table gives an action or output when a state or input comes in. The action is selected by the maximum of the Q-values in that state. For example, if the robot detects a range of 2.1, an action that will be selected is going right because 0.91 is the maximum number of the second row. The Q-learning equation is shown in Equation 2.1. If it is the last step or reaching the goal, the new Q-value is equal to the reward (see Equation 2.2).

Table 2.3

An Example of a Q-Table

Q(s, a)	Go Up	Go Down	Go Left	Go Right
[0, 2]	0.23	0.57	0.82	0.11
(2, 4]	0.15	0.08	0.51	0.91
(4, 6]	0.92	0.52	0.22	0.02
> 6	0.17	1.22	0.75	0.59

Note. The Q-table represents Q-values for states (s) and actions (a). This table is an example of an environment. The states are the range of a sensor or some information, and the actions are the direction of what it should go.

The Q-learning equation that is not the last step is:

$$Q(s,a)_{new} = Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a)) \quad (2.1)$$

Where, $Q(s,a)_{new}$ is a new Q-value of state and action that is changed,

$Q(s,a)$ is an old Q-value of state and action that is changed,

α is a learning rate or alpha,

r is a reward for a step,

γ is a discount factor or gamma,

$max_{a'} Q(s',a')$ is a maximum of old Q-value of a next state.

The Q-learning equation that is the last step is:

$$Q(s,a)_{new} = r \tag{2.2}$$

The training algorithm steps of Q-learning are as follows:

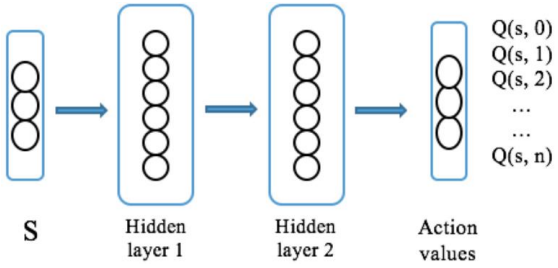
1. Design a Q-table, and initialize an epsilon to 1.
2. Random a number between 0 and 1, and if the number is less than or equal to the epsilon, the action will be random. If the number is greater than the epsilon, the action will be picked from the Q-table.
3. Do the action that it gets from number 2, and it will give a new state.
4. Update Q-table from Equation 2.1 or 2.2.
5. Reduce the epsilon, and repeat number 2.

2.2.3 Deep Q-Learning

From Table 2.3, if the state consists of many sensors and many states of each sensor or an image, it is hard to find all the Q-values of those states and actions, and it will not be possible to find all the Q-values if the state is an image. Thus, deep q-learning was created to solve this problem. A Q-table is replaced by a neural network. The input of the neural network is a state, and the output is q-values that will provide an action. The neural network is called a deep Q-network or DQN (see Figure 2.12).

Figure 2.12

An example of a DQN



Note. If an input is an image, a CNN can be applied at the first layer.

The training algorithm for deep Q-learning is the same as Q-learning, but the getting value from the Q-table is replaced by prediction of DQN, and the updating of the Q-table is replaced by updating or training of DQN.

2.2.4 Double Deep Q-Learning

From deep Q-learning, when it calculates Q-values of the current state and Q-values of the next state, it uses the same neural network. It causes a divergence because the Q-values of the next state keep on changing, and the Q-values of the current state will chase the Q-values of the next state that is always changing.

The problem will be solved by using double DQN. The Double DQN method uses two neural networks. The first neural network is used for calculating Q-values of the current state, and this network is always changing by training or updating it. The second neural network is used for calculating Q-values of the next state, and this network is changed every T step, where T is a hyperparameter. The second network is changed by copying the first neural network.

2.2.5 Experience Replay

In DQN and double DQN, having one current step is not stable for training neural networks because the neural networks are always changing. The experience replay algorithm is used to improve stability. It stores every step in replay memory. The replay memory is limited by a number. When the replay memory is full, the new one will replace the oldest one in the replay memory. When in the training step, it will randomly pick in replay memories by a number that is more than one. It is not trained by the state of the current step.

2.3 Simulator

When reinforcement learning is used in one project, there is always a simulator for training reinforcement learning networks. For autonomous driving, there are many simulators.

2.3.1 Related Work

When reinforcement learning is used in one project, there is always a simulator for training reinforcement learning networks. For autonomous driving, there are many simulators.

Samuel Arzt presented deep learning cars using Unity in two-dimensional as shown in Figure 2.13. Unity is a game engine. He used Unity to train his reinforcement learning network. When he uses a game engine, he has to build a game by himself because it is not ready to use.

Abdur Razzaq Fayjie, Sabir Hossain, Doukhi Oualid, and Deok Jin Lee presented driverless cars using Unity to simulate the urban environment in three-dimensional space, as shown in Figure 2.14.

Figure 2.13

Driving Cars on Unity Game Engine in a Two-Dimensional Simulator

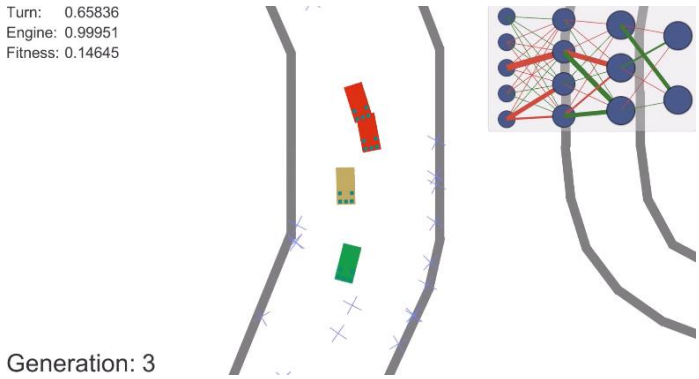


Figure 2.14

Driving Cars on Unity Game Engine in a Three-Dimensional Simulator

Takafumi Okuyama, Tad Gonsalves, and Jaychand Upadhay presented an autonomous driving system using Unity to simulate straight road boundaries and obstacles in three-dimensional.

Syed Owais Ali Chishti, Sana Riaz, Muhammad Bilal Zaib and Mohammad Nauman presented self-driving cars in a real environment. This method has a disadvantage, which is the training time because it was not trained in parallel. There is only one agent at one time, as shown in Figure 2.15.

Harrison Kinsley presented driving cars in a video game called Grand Theft Auto V. The advantage of this method is that the environment in the game looks real, but using a video game that is always generating high graphics cannot be trained by parallel as shown in Figure 2.16.

Figure 2.15

Driving Cars in the Real Environment

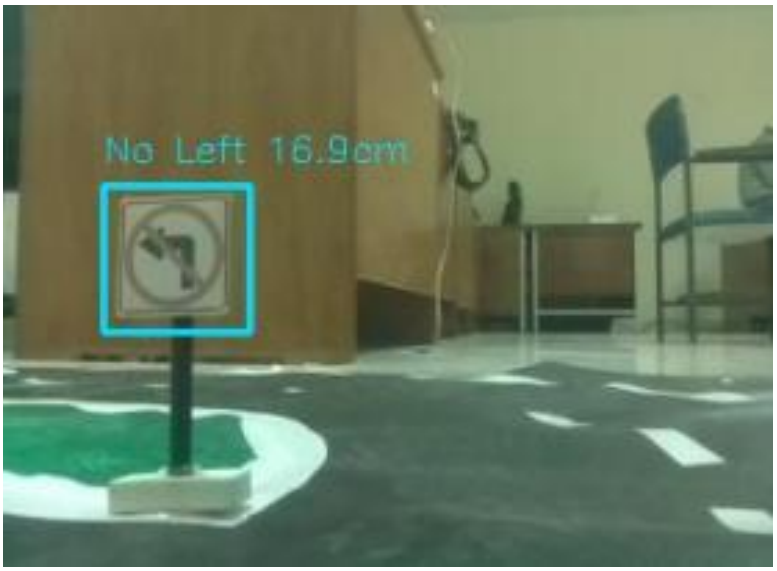


Figure 2.16

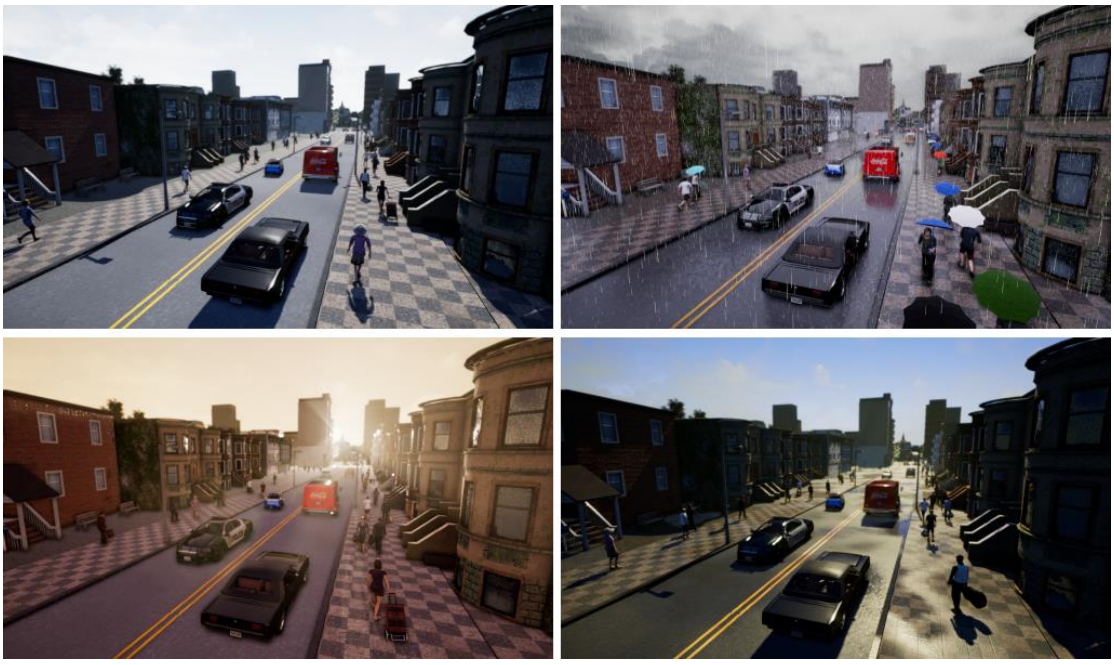
Driving Cars in Grand Theft Auto V

2.3.2 CARLA Simulator

CARLA is one of the driving simulators for autonomous driving. It was built by Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. It is based on the Unreal Engine 4. CARLA has been developed from the ground up to support the development, training, and validation of autonomous urban driving systems. CARLA provides open digital assets (urban layouts, buildings, and vehicles) that were created for this purpose and can be used freely. From Figure 2.17, CARLA can change weather conditions. Changing weather conditions will improve training. It will make a variety of training sets. From Figure 2.18, CARLA has a semantic segmentation function. This function will be useful for training image segmentation networks before data gets into the deep q-network. CARLA can also run without rendering graphics.

Figure 2.17

Changing of Four Weather Conditions in CARLA



Harrison Kinsley used CARLA for training self-driving cars using a reinforcement network. He coded the network using Python and used the Python API to control CARLA. Figure 2.19 shows the result of a self-driving car using CARLA. He could make cars stay in lanes, and the cars could change lanes.

Figure 2.18

Semantic Segmentation Function in CARLA

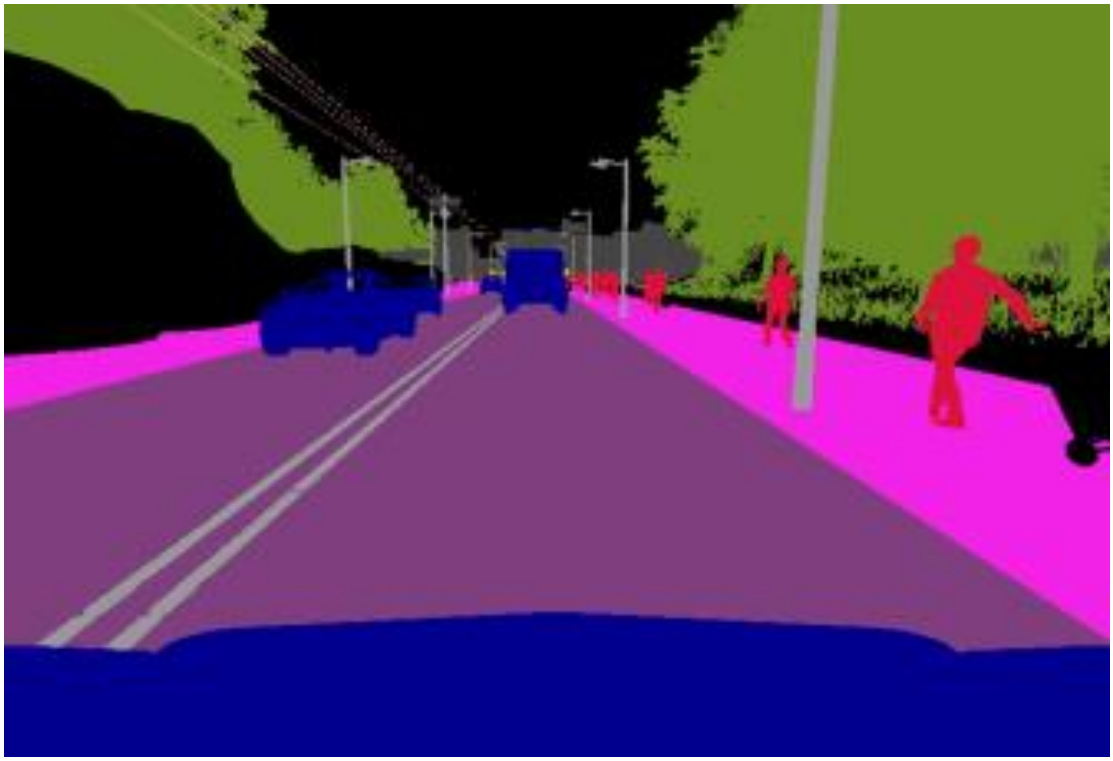
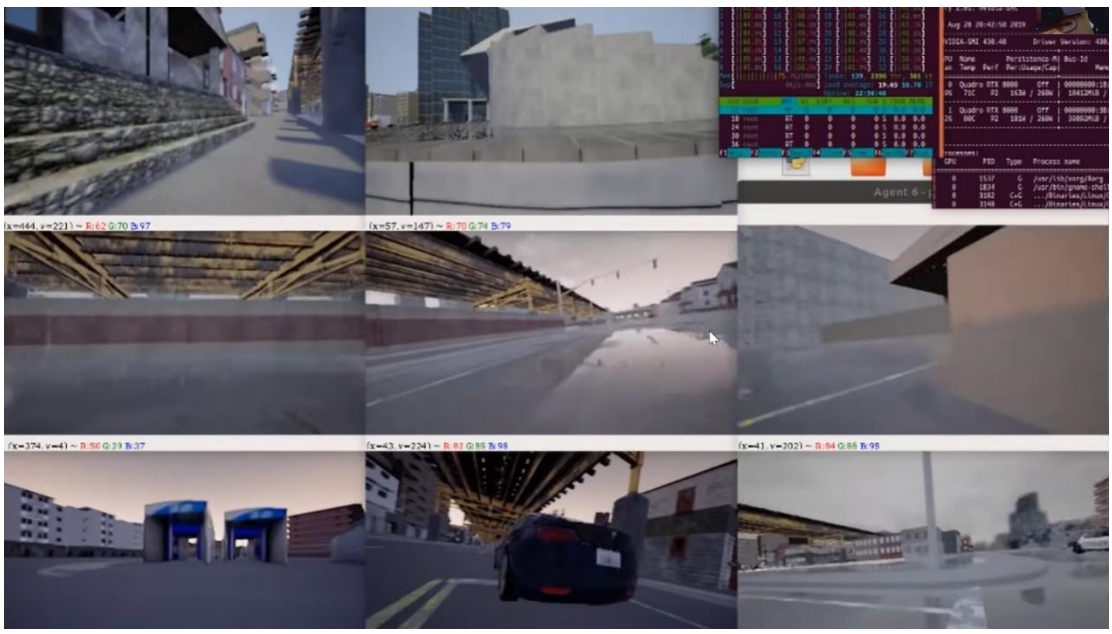


Figure 2.19

Showing of Self-Driving Car Using CARLA



CHAPTER 3
METHODOLOGY

3.1 Overview

This thesis uses only one sensor for controlling a car. The sensor is a camera. The main control algorithm is a double DQN. It learns to drive a car in a simulator. The CARLA simulator was selected to be used in this thesis. Because when training double DQN in a simulator, the network needs the same input image between the simulator and the real world. Semantic segmentation is selected to convert images to make them the same. The real-world images are converted to segmented images, then segmented images are fed to a double DQN to predict an action to control a golf cart.

3.2 Equipment Selection

3.2.1 Golf Cart

A model of golf cart that is used in this thesis is the EVT Echo 4s (see Figure 3.1). The EVT Echo 4s is an electric golf cart. A specification is shown in Table 3.1.

Figure 3.1

A Golf Cart Model EVT Echo 4s

Table 3.1

A specification for EVT Echo 4s

Maximum Speed	45 kilometers per hour
Turning Radius	5 meters
Overall Length	3160 millimeters
Overall Width	1220 millimeters
Overall Height	1830 millimeters
Wheelbase Dimension	2410 millimeters
Tire Size	205/50 R10

The maximum speed is limited at 8 kilometers per hour because the golf cart easily stops with this speed limit when out of control the golf cart when testing.

The golf cart can be controlled by a digital signal via a USB serial port. It can control the position of the steering wheel from far-left to far-right, the level of the throttle paddle from 0 percent to 100 percent, and the level of the brake paddle from 0 percent to 100 percent. USB sends serial to the golf cart with a bytes array [36, position of steering wheel, level of throttle paddle, level of brake paddle, 64] which are all decimal numbers. The first and last position of the array is the head and tail value. The position of the steering wheel is in the range of 0 to 255, where 0 means the far-left position, 255 means the far-right position, and the middle position is 128. The level of the throttle paddle and brake paddle is in the range of 0 to 255, which is converted from the range of 0 to 100. The bytes array is converted from decimal numbers to hexadecimal numbers before being sent to the golf cart. The golf cart can receive a command to change an action every 400 milliseconds.

3.2.2 Computing Hardware

The first computer was a desktop. The specifications of this desktop are an Intel I3-8100 CPU, a GTX 1660 Ti GPU, 24 gigabytes of RAM, and Windows 10 OS. It is for training a double DQN.

The second computer is a desktop. The specifications of this desktop are a Ryzen 5-2600 CPU, a GTX 1050 Ti GPU, 16 gigabytes of RAM, and Windows 10 OS. It is for running a simulator when training a double DQN. The first computer and the second computer cannot train a double DQN and run a simulator at the same time on a single computer due to a GPU's out of memory.

The third computer is a laptop. The specifications of this laptop are an Intel I7-9750H CPU, a RTX 2070 GPU, 32 gigabytes of RAM, and Windows 10 OS. It is for training a semantic segmentation network and running a double DQN to control a golf cart in a real-world experiment.

3.2.3 Camera

This thesis uses a webcam. A model of this webcam is the Logitech C525 (see Figure 3.2). A specification is shown in Table 3.2.

From Section 4.1, the resolution of the camera is fixed at 864x480 pixels, then it is cropped and resized to 512x512 pixels, and the frame rate is fixed at 5 fps due to the system performance (see Section 4.3.2.1).

The camera is installed on the golf cart at a height of 1675 millimeters from the ground, 450 millimeters from the front of the golf cart, and 77.5 degrees to the ground. These positions were roughly measured by a tape measure.

Figure 3.2

Logitech C525

Table 3.2

A specification for the Logitech C525

Resolution	1280x720 pixels
Frame Rate	30 frames per second
Field of View	69 degrees

3.3 Semantic Segmentaion

3.3.1 Model Selection

From Table 2.2, it shows the HRNet-OCR model is the best choice for semantic segmentation, but it cannot be trained on my computers due to GPU's out of memory (see Section 4.2.1).

Pavel Yakubovskiy, or username Qubvel on GitHub, created a high-level API for segmentation models based on Pytorch. I decided to use this API because it took less time when compared with using source codes. Most of the source codes were created for training with mixed-precision training that is available only on Ubuntu OS. All my computers are Windows 10 OS. It took a long time to modify their codes. This Qubvel's API consists of DeepLabv3 and PSPNet. These two models have high scores on the Cityscapes test dataset (see Table 2.2). The PSPNet was chosen for this thesis because the DeepLabv3 takes more time than the PSPNet (see Section 4.2.2 and 4.2.3).

3.3.2 Dataset

There are 2 semantic segmentation models in this thesis. The first model is used to segment real images, and the second model is used to segment images in the CARLA simulator. The dataset that was used in this thesis is shown in Table 3.3. The first model was trained by the AIT dataset, the augmented AIT dataset, and the Mapillary Vistas dataset. The second model was trained by the CARLA dataset.

The reason for adding the augmented AIT dataset and the Mapillary Vistas dataset is to avoid dataset overfitting of the main dataset.

The AIT dataset was recorded by a front-view camera. The images were labeled into 5 classes, which are roads, lane markings, vehicles, humans, and others (see Figure 3.3).

Table 3.3

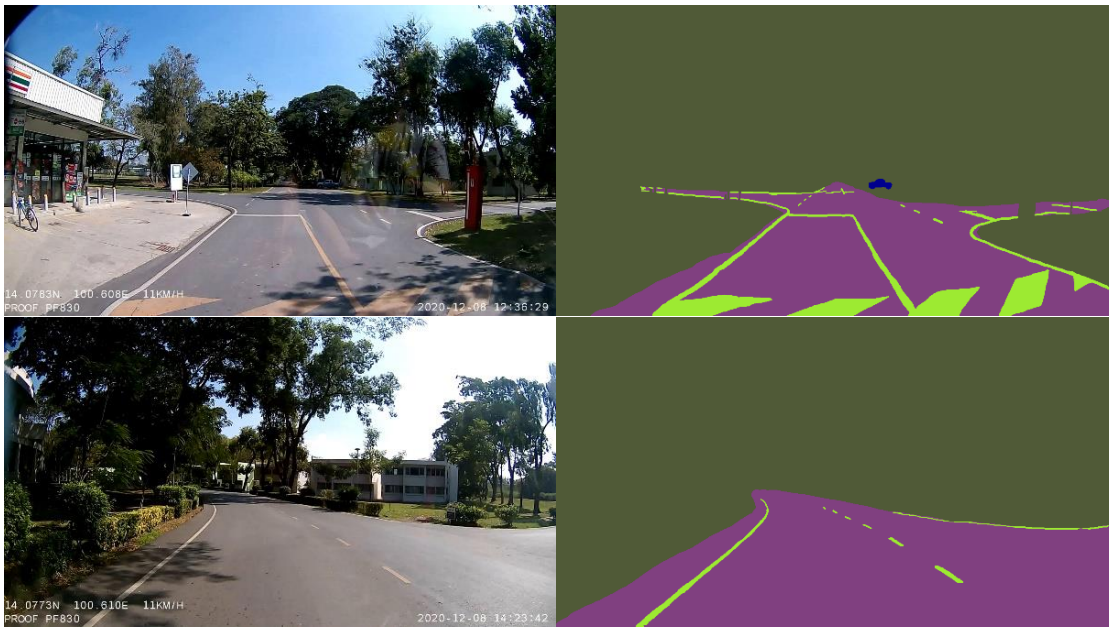
The Number of Images in Each Dataset

Name	Training Dataset	Validation Dataset	All
AIT	1607	689	2296
Augmented AIT	11249	0	11249
Mapillary Vistas	3000	0	3000
CARLA	1121	481	1602

Note. It was split 30 percent for the validation set on the AIT and the CARLA dataset. The Augmented AIT and the Mapillary Vistas do not require to split for validation set because they are not the main dataset for use.

Figure 3.3

Example of the AIT Dataset



The augmented AIT dataset was augmented from the AIT training set with 7 times which it was randomly cropped, rotated, flipped, blurred, and added Gaussian noise (see Figure 3.4).

Figure 3.4

Example of the Augmented AIT Dataset

The Mapillary Vistas dataset is a public dataset that is free to use. There are 20000 images in the training set. They were randomly selected from 3000 images. There are 124 classes in the Mapillary Vistas dataset. The 124 classes were converted into 5 classes, which are roads, lane markings, vehicles, humans, and others. The original is shown in Figure 3.5.

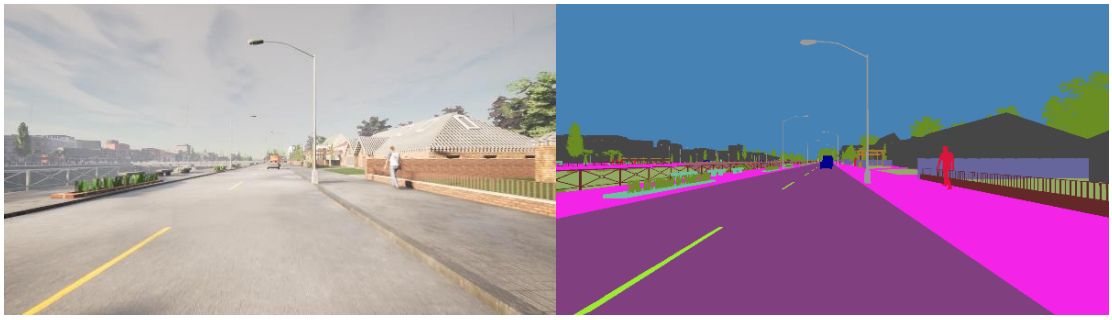
Figure 3.5

Example of the Mapillary Vistas Dataset

The CARLA dataset was recorded in the simulator. The CARLA dataset has 24 different classes. The 24 classifications were reduced to 5, which are roads, lane markers, cars, humans, and others. Figure 3.6 represents the original.

Figure 3.6

Example of the CARLA Dataset



3.3.3 Preprocessing

The normalization method was selected to preprocess images as shown in Equation 3.1. Normalization is used after receiving an image from the camera. Mean and standard values were found from the AIT dataset for each color channel. The values are shown in Section 4.2.3.

The normalization equation is:

$$X_{Norm} = \frac{X - mean(x)}{std(x)} \tag{3.1}$$

3.3.4 Training

The PSPNet experiment 1 and 2 is to segment real images. The PSPNet experiment 3 is to segment images in the CARLA simulator.

3.3.4.1 **PSPNet Experiment 1.** This experiment was created to train the PSPNet. The image size is 272x272. It was trained using the AIT dataset, the augmented AIT dataset, and the Mapillary Vistas dataset. Hyperparameters are shown in Table 3.4. The result is shown in Section 4.2.5.1.

3.3.4.2 **PSPNet Experiment 2.** The trained model from experiment 1 provides poor quality segmented images, especially the lane markings. This experiment wants to improve them by increasing the image size from 272x272 to 512x512. The purpose of this experiment was to train the PSPNet. It was trained using the AIT dataset, the augmented AIT dataset, and the Mapillary Vistas dataset. Hyperparameters are shown in Table 3.5. The result is shown in Section 4.2.5.2.

Table 3.4

Hyperparameters of Semantic Segmentation Experiment 1

Hyperparameter	Value
Loss Function	Cross Entropy
Optimization	Adam
Learning Rate	0.0001
Batch Size	32

Table 3.5

Hyperparameters of Semantic Segmentation Experiment 2 and 3

Hyperparameter	Value
Loss Function	Cross Entropy
Optimization	Adam
Learning Rate	0.0001
Batch Size	8

3.3.4.3 **PSPNet Experiment 3.** This experiment was created to want a model to use in the CARLA simulator. The model from experiment 2 cannot be used because the model has never seen images in the CARLA simulator before, but the model can be used as a pre-trained model for this experiment. The image size is 512x512. It was trained using the CARLA dataset. Hyperparameters are shown in Table 3.4, which is the same as experiment 2. The result is shown in Section 4.2.5.3.

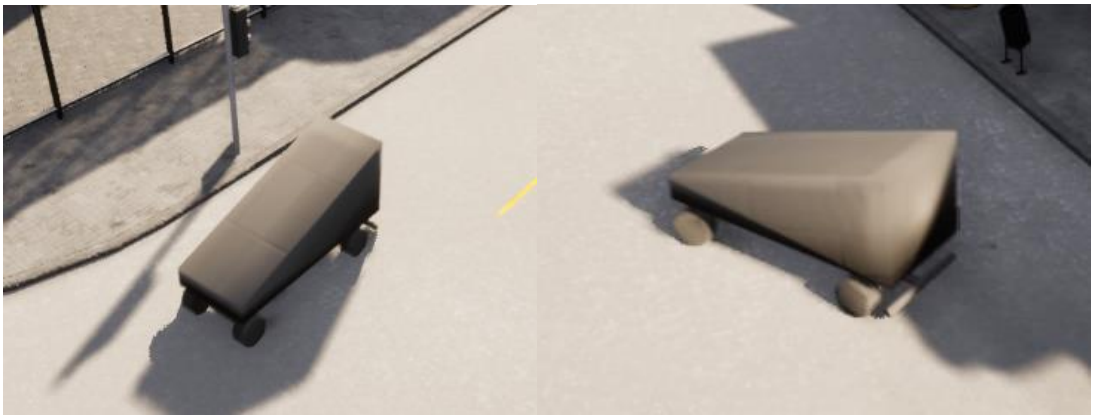
3.4 CARLA Simulator

3.4.1 Creating A Golf Cart Model

A golf cart model was created to make it the same spec as the real golf cart (see Figure 3.7). It got the reference from Table 3.1. It was created using the Blender program, which is a free program for creating three dimensional models.

Figure 3.7

Golf Cart Model in CARLA Simulator

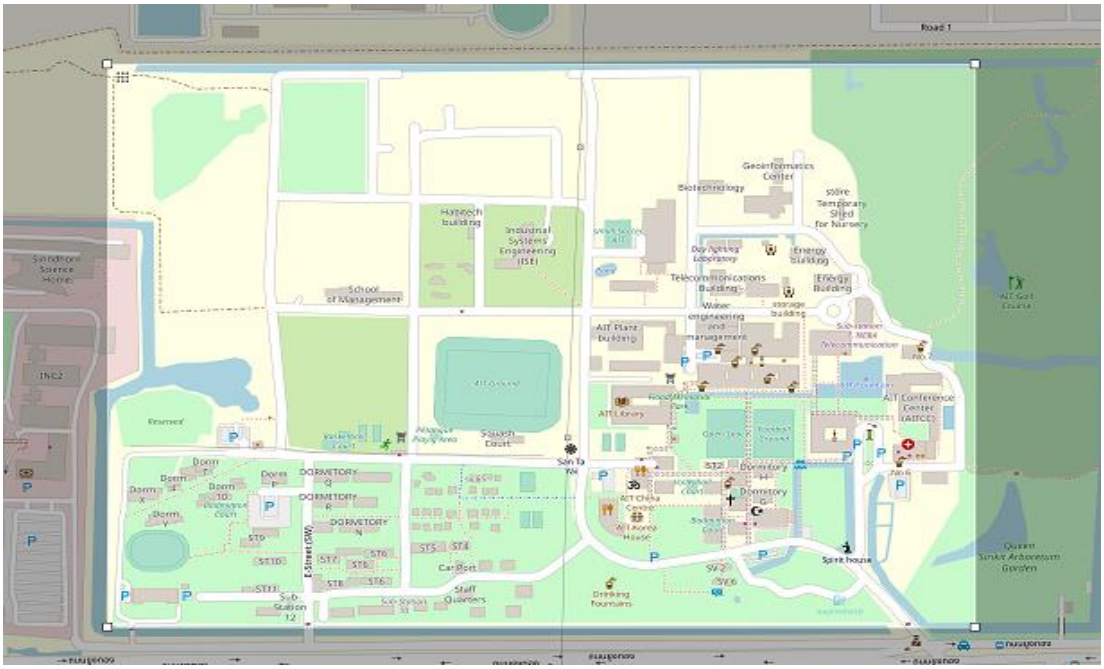


3.4.2 Creating the AIT Map

1. Grabbing the AIT Map data from OpenStreetMap as shown in Figure 3.8.

Figure 3.8

AIT Map from OpenStreetMap



2. Importing from the OpenStreetMap to the RoadRunner program. It imported only a guideline with the real scale as shown in Figure 3.9.

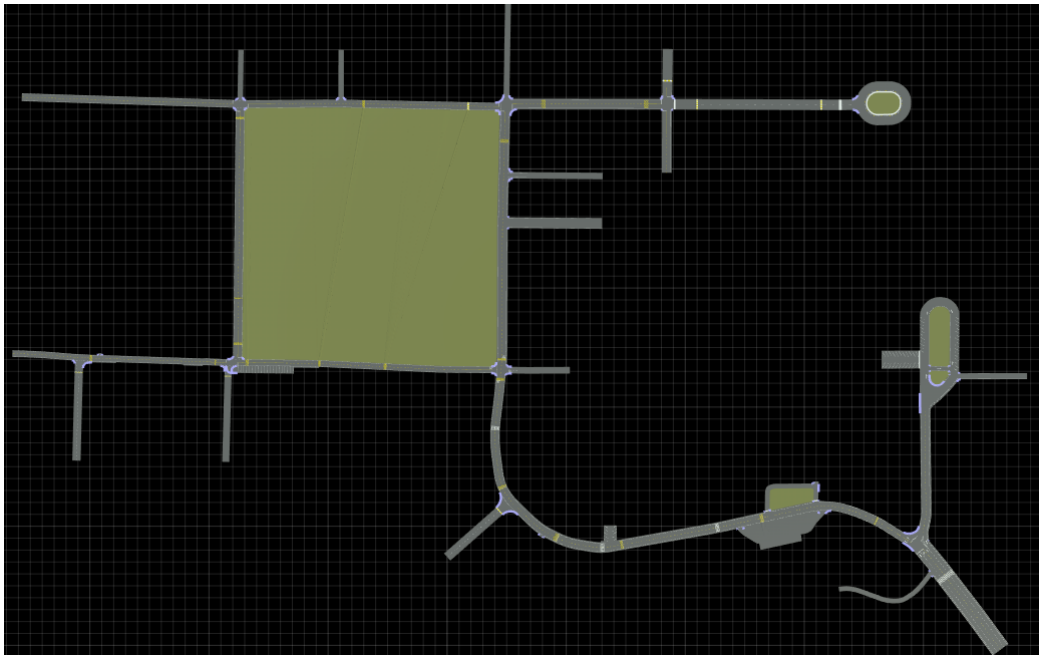
Figure 3.9

Guideline from OpenStreetMap

3. Drawing roads using the RoadRunner program (see Figure 3.10).

Figure 3.10

Finished Drawing Roads



4. Importing the map from the RoadRunner to CARLA Simulator, and add objects as shown in Figure 3.11 and 3.12.

Figure 3.11

Add Objects to the Map

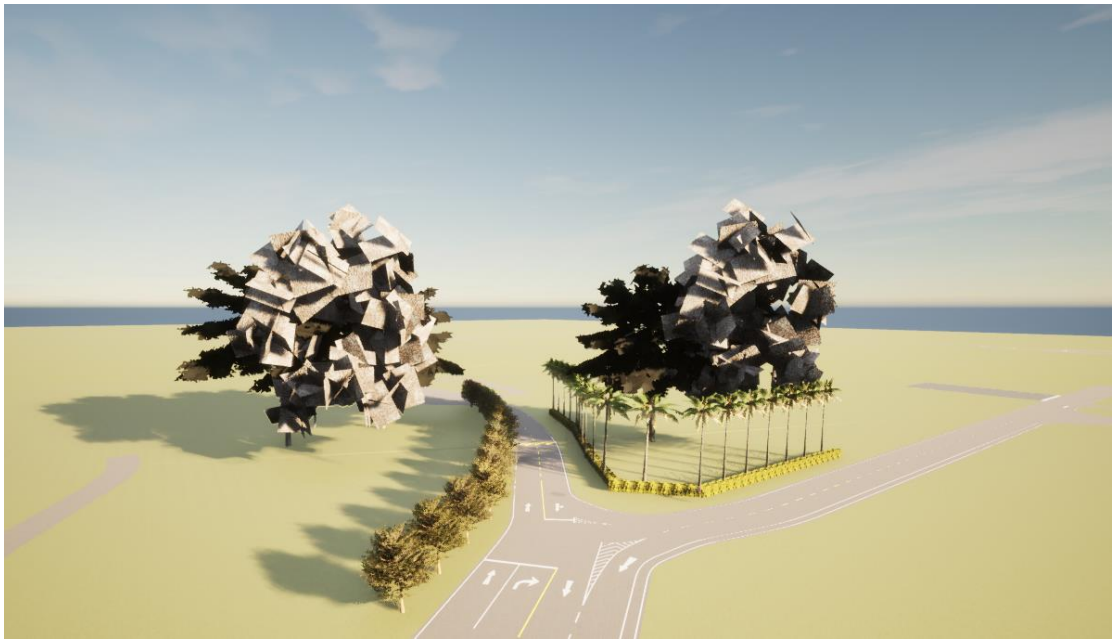
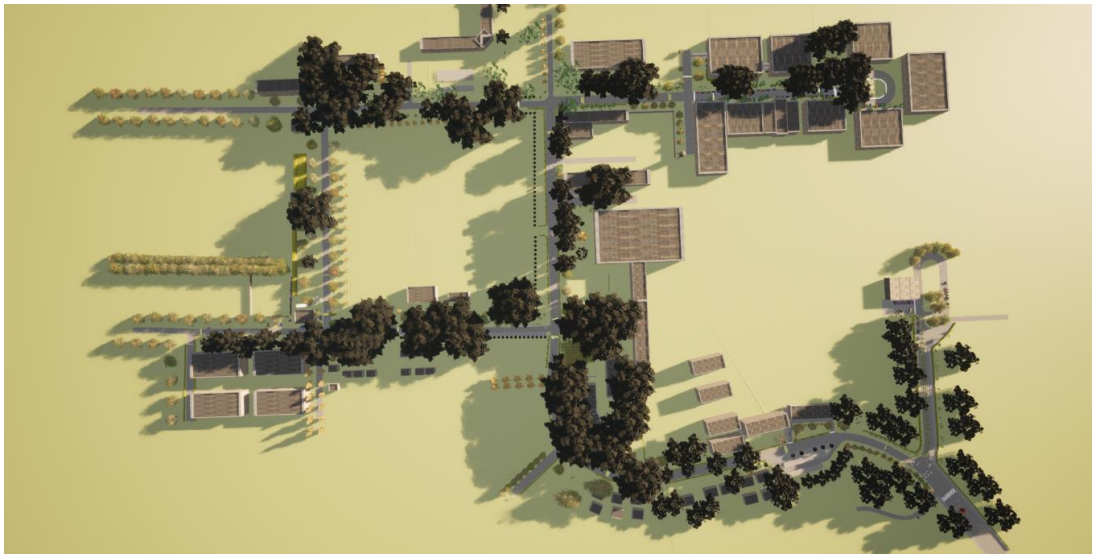


Figure 3.12

Finished Adding Objects to the Map



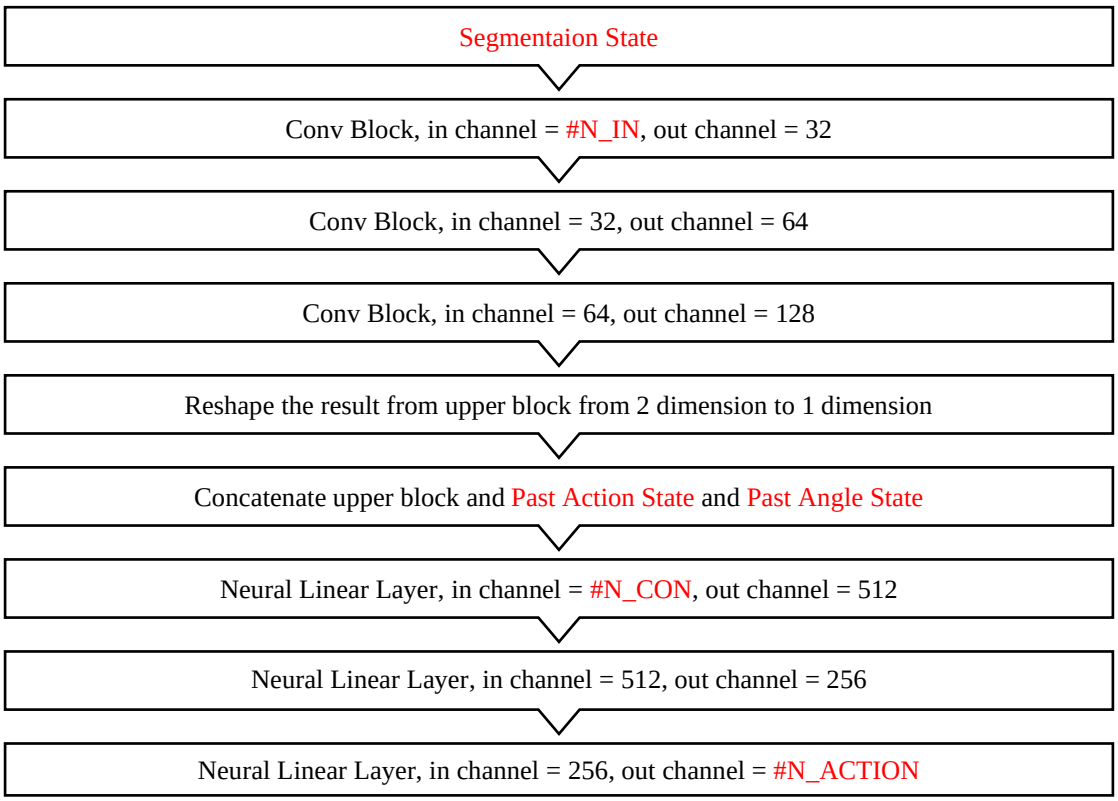
3.5 Double Deep Q-Learning

3.5.1 Deep Q-Network Architecture

The deep Q-network architecture is shown in Figure 3.13. There are Conv Blocks inside of the deep Q-network, which is shown in Figure 3.14.

Figure 3.13

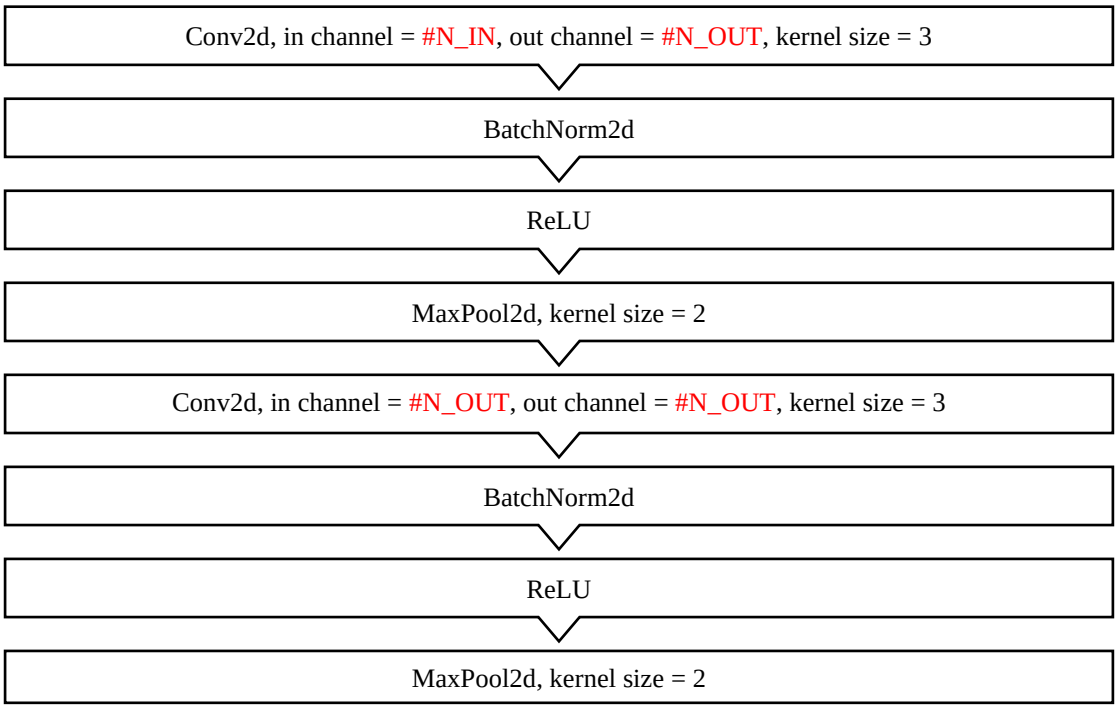
Deep Q-Network Architecture



Note. There are 3 inputs, which are Segmentation State, Past Action State, and Past Angle State. The Segmentation State contains previous segmentation images. The number of previous segmentation images depends on each experiment. The #N_IN is the number of all segmentation images. The Past Action State contains the past actions. The Past Angle State contains the past level of the throttle paddle, the angle of the steering wheel, and the level of the brake paddle. The number of past actions and past angles depends on each experiment. The #N_CON is the number after the concatenation. The #N_ACTION depends on each experiment. The Conv Block is shown in Figure 3.14.

Figure 3.14

Conv Block



3.5.2 Training Scene

There are 3 training scenes as shown in Figures 3.15 to 3.17. The goal is to move from point A to B in each scene.

Figure 3.15

Training Scene 1

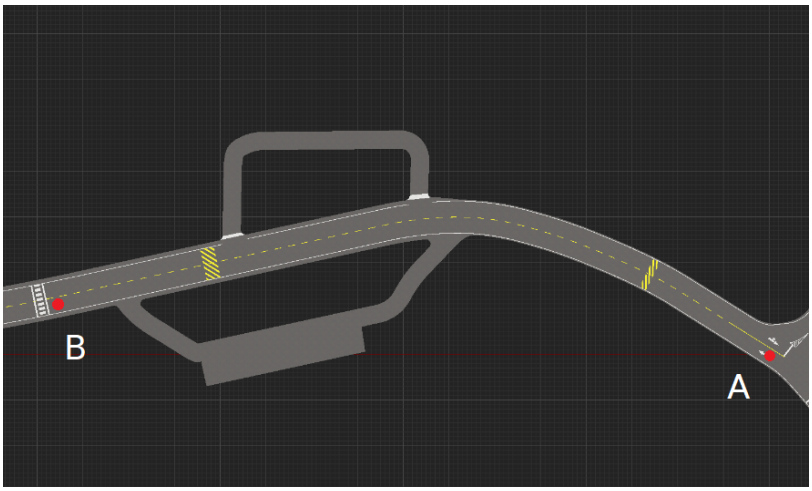


Figure 3.16

Training Scene 2

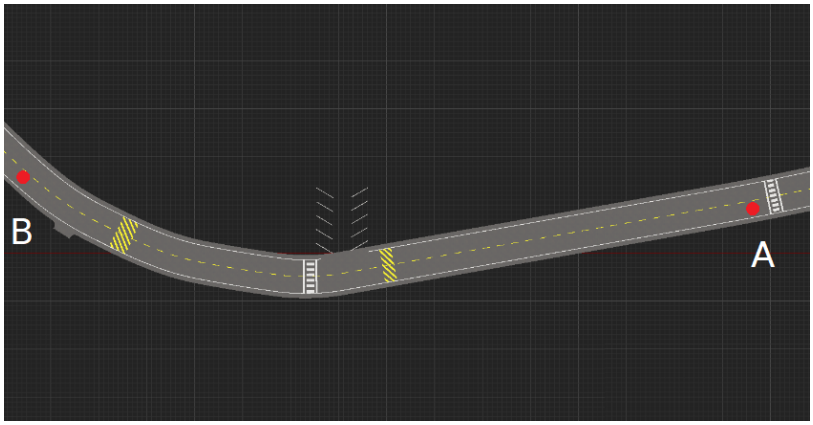
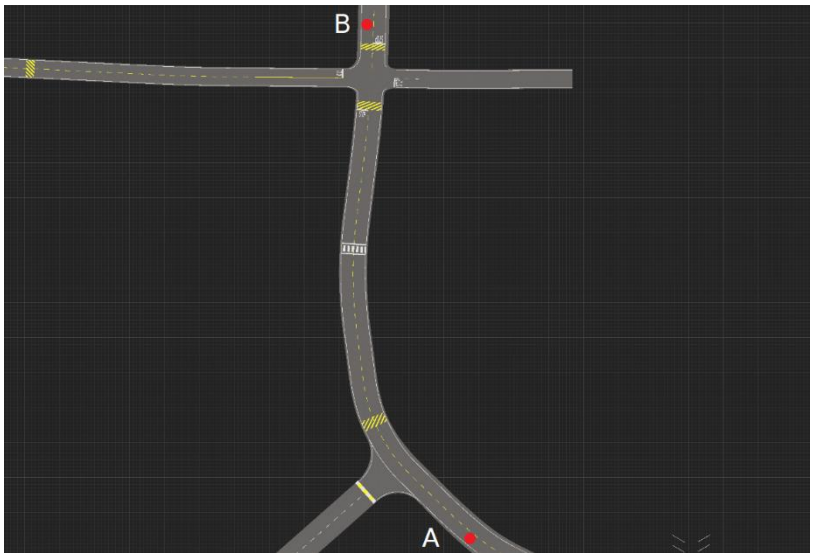


Figure 3.17

Training Scene 3



3.5.3 Training

3.5.3.1 **Double DQN Experiment 1.** This experiment used training scene 2 as shown in Figure 3.16. The position of the golf cart was spawned for 5 points as shown in Figure 3.18. This experiment used segmented images that CARLA provided (see Figure 2.18) as the input to DQN. The purpose of this experiment is to test the algorithm. There are 8 actions as shown in Table 3.6. A reward was calculated using

Equation 3.2. If the golf cart is out of the road or crashes into something, the reward will be -2. Hyperparameters are shown in Table 3.7, and hyperparameters for only this experiment are shown in Table 3.8. The result is shown in Section 4.3.1.1.

Figure 3.18

5 Position of Spawning the Golf Cart in Training Scene 2

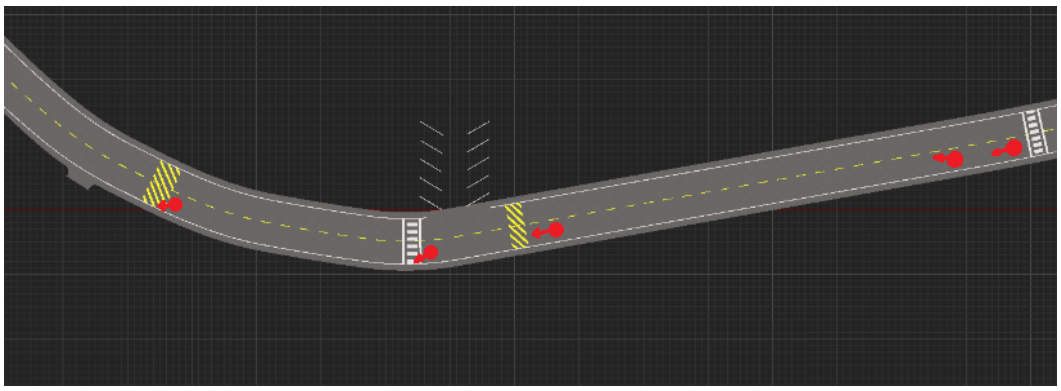


Table 3.6

Actions of Double DQN Experiment 1 and 2

Action Number	Action	Comment
0	T-0.0667, if S<0 S+0.0667 if S>0 S-0.0667, B-0.0667	Release
1	T+0.1, if S<0 S+0.0667 if S>0 S-0.0667, B=0	Forward
2	T-0.0667, S-0.1, B-0.0667	Left
3	T-0.0667, S+0.1, B-0.0667	Right
4	T=0, if S<0 S+0.0667 if S>0 S-0.0667, B+0.1	Brake
5	T=0, if S<0 S+0.0667 if S>0 S-0.0667, B=1	Stop
6	T+0.1, S-0.1, B=0	Forward&Left
7	T+0.1, S+0.1, B=0	Forward&Right

Note. T means level of throttle paddle. S means angle of steering wheel. B means level of break paddle.

The reward equation for every double DQN experiment is:

$$r = \frac{current\ speed}{\max speed} + \frac{area\ of\ car\ in\ left\ lane}{area\ of\ car} \quad (3.2)$$

Table 3.7

Hyperparameters of Every Double DQN Experiment

Hyperparameters	Values
Learning Rate	0.0001
Discount Factor	0.9
Target Update Interval	1000 steps
Play Interval	900 steps
Epsilon	1
Epsilon Decay Rate	0.99999
Minimum Epsilon	0.1
Maximum Replay Memory	100000
Replay Size	32

Table 3.8

Hyperparameters of Double DQN Experiment 1

Hyperparameters	Values
Number of Input Segmented Images	14 frames
Number of Past Actions	20
Number of Past Angles	20
Max Speed	20 km/h
Frame Time	0.1 seconds
Action Time	0.2 seconds

***3.5.3.2* Double DQN Experiment 2.** This experiment used segmented images from the PSPNet. The position of the camera was changed to the same as the position of the camera on the real golf cart (see Section 3.2.3). The action time was changed from 0.2 seconds to 0.4 seconds because the golf cart can be sent a command every 0.4 seconds (see Section 3.2.1). The rest of the setting was the same as experiment 1, which were training on scene 2, the golf cart was spawned for 5 points as shown in Figure 3.18, using 8 actions is shown in Table 3.6, a reward was calculated using Equation 3.2, and if the golf cart is out of the road or crashes into something, the reward will be -2. The reason for using the same setting as in experiment 1 is that the setting gives acceptable results. The new hyperparameters are shown in Table 3.9. The result is shown in Section 4.3.1.2.

Table 3.9

Hyperparameters of Double DQN Experiment 2

Hyperparameters	Values
Number of Input Segmented Images	14 frames
Number of Past Actions	20
Number of Past Angles	20
Max Speed	20 km/h
Frame Time	0.1 seconds
Action Time	0.4 seconds

***3.5.3.3* Double DQN Experiment 3.** This experiment was trained using training scenes 1, 2, and 3 from Section 3.5.2, and used only 1 spawn point, which is point A in each scene. If it was trained using 5 spawn points for 3 scenes, it would take a long time to train. The maximum speed of the golf cart was changed from 20 kilometers per hour to 8 kilometers per hour because 20 kilometers per hour is too fast for testing. The driver will not be able to stop the golf cart if it is out of control. When changing the maximum speed, actions need to change because the actions from experiments 1 and 2 will decrease the rate of change reward. The DQN will not know if it changed an action.

That action will give a reward more or less. The new actions are shown in Table 3.10, and the new hyperparameters are shown in Table 3.11. The rest of the setting was the same as experiment 2, which used the PSPNet, a reward was calculated using Equation 3.2, and if the golf cart is out of the road or crashes into something, the reward will be -2. The result is shown in Section 4.3.1.3.

Table 3.10

Actions of Double DQN Experiment 3

Action Number	Action	Comment
0	T-0.2, if S<0 S+0.1333 if S>0 S-0.1333, B-0.2	Release
1	T+0.2, if S<0 S+0.1333 if S>0 S-0.1333, B=0	Forward
2	T-0.2, S-0.1333, B-0.2	Left
3	T-0.2, S+0.1333, B-0.2	Right
4	T=0, if S<0 S+0.1333 if S>0 S-0.1333, B+0.2	Brake
5	T=0, if S<0 S+0.1333 if S>0 S-0.1333, B=1	Stop
6	T+0.2, S-0.1333, B=0	Forward&Left
7	T+0.2, S+0.1333, B=0	Forward&Right

***3.5.3.4* Double DQN Experiment 4.** This experiment wants to try a new way to give the reward by adding deducting 1 when the golf cart goes outside the left lane because I want to make the golf cart focus on staying in the left lane. If the area of the car in the left lane divided by the area of the car is more than or equal to 0.98, the reward will be calculated using the same equation as previously, which is Equation 3.2. If the area of the car in the left lane divided by the area of the car is less than 0.98, the reward will be calculated using the new equation, which is Equation 3.3, and if the golf cart is out of the road or crashes into something, the reward will be -4, which was changed from -2. The actions were decreased from 8 actions to 3 actions because I want to improve the speed of training. Driving in the left lane requires only 3 actions. When testing the golf cart with the model from experiment 3, it took a long time to control the

steering wheel from the left position to the right position. The new actions are shown in Table 3.12. The rest of the setting was the same as experiment 3, which used the PSPNet, it was trained using training scenes 1, 2, and 3, hyperparameters are shown in Table 3.11. The result is shown in Section 4.3.1.4.

Table 3.11

Hyperparameters of Double DQN Experiment 3 and 4

Hyperparameters	Values
Number of Input Segmented Images	14 frames
Number of Past Actions	20
Number of Past Angles	20
Max Speed	8 km/h
Frame Time	0.1 second
Action Time	0.4 second

The reward equation for the double DQN experiment 4 and 5, if the area of the car in the left lane divided by the area of the car is less than 0.98, is:

$$r = \frac{current\ speed}{\max speed} + \frac{area\ of\ car\ in\ left\ lane}{area\ of\ car} - 1 \quad (3.3)$$

Table 3.12

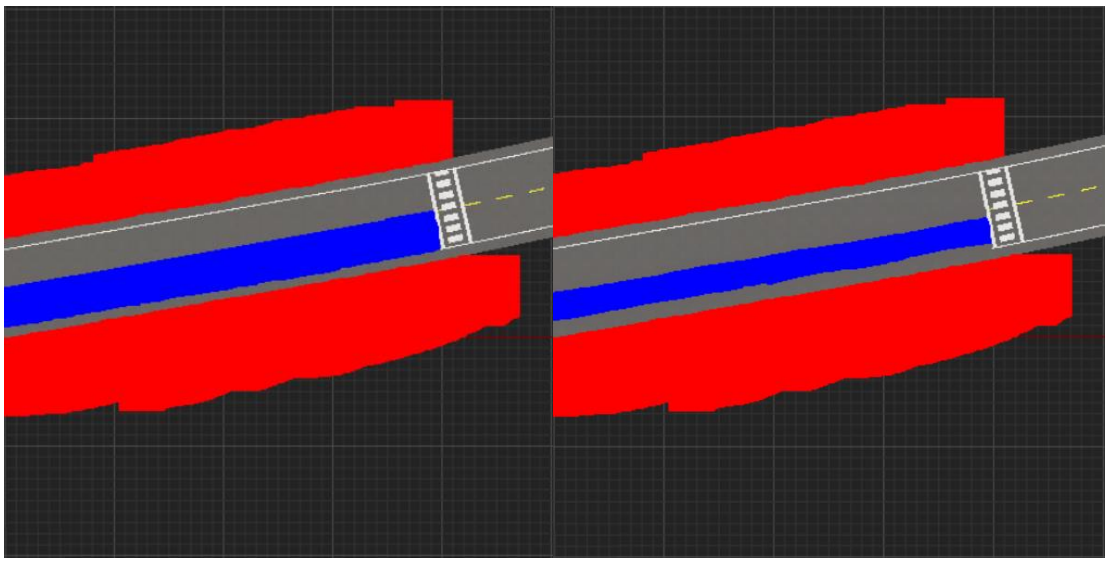
Actions of Double DQN Experiment 4 and 5

Action Number	Action	Comment
0	T+0.2, if S<0 S+0.2 if S>0 S-0.2, B=0	Forward
1	T+0.2, if S<0 S-0.1 if S>0 S-0.4, B=0	Forward+Left
2	T+0.2, if S<0 S+0.4 if S>0 S+0.1, B=0	Forward+Right

***3.5.3.5* Double DQN Experiment 5.** When testing the golf cart with the model from experiment 3, the steering wheel was controlled when the golf cart was close to the lane marking. In this experiment, the left lane of the road was decreased the width to make it drive in the center (see Figure 3.19). There was a time peak more than 0.1 seconds when testing with the real golf cart from experiment 3, as shown in Section 4.3.2.1. Thus, this experiment was increased the frame time from 0.1 seconds to 0.2 seconds, as shown in Table 3.13. There was a bit of change in giving a reward from experiment 4. There was too much control over the steering wheel. To fix that problem, the reward will be deducted by 0.2 when moving the steering wheel. The rest of the setting was the same as experiment 4, which was giving a reward, if the area of the car in the left lane divided by the area of the car is more than or equal to 0.98, the reward will be calculated using Equation 3.2. If the area of the car in the left lane divided by the area of the car is less than 0.98, the reward will be calculated using Equation 3.3, if the golf cart is out of the road or crashes into something, the reward will be -4, the PSPNet was used, it was trained using training scenes 1, 2, and 3, and the actions were shown in Table 3.12. The result is shown in Section 4.3.1.5.

Figure 3.19

Changing Reward Mask for Double DQN Experiment 5



(a) The old reward mask (b) The new reward mask

Table 3.13

Hyperparameters of Double DQN Experiment 5

Hyperparameters	Values
Number of Input Segmented Images	15 frames
Number of Past Actions	8
Number of Past Angles	8
Max Speed	8 km/h
Frame Time	0.2 seconds
Action Time	0.4 seconds

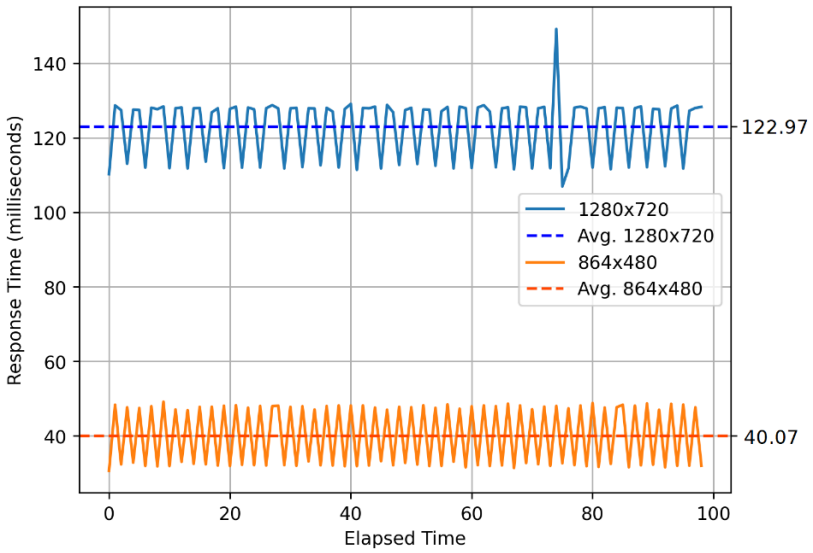
CHAPTER 4
RESULTS

4.1 Camera Setting

From Section 4.2.5.2, the resolution that makes the PSPNet get the best quality is 512x512. The ratio of the AIT dataset is 16:9. The resolution of the Logitech C525 can be set by OpenCV at 1280x720 or 864x480, where the ratio of 864x480 is not 16:9. It should be 853x480. It can be cropped from the center. This thesis chooses a resolution of 864x480 because it takes a shorter response time than 1280x720, as shown in Figure 4.1. It was tested on my laptop that will be used when testing with the real golf cart. The average response time of 1280x720 is 122.97 milliseconds or 8.13 frames per second. The average response time of 864x480 is 40.07 milliseconds or 24.96 frames per second. The resolution of 864x480 was chosen to be used in this thesis because it took less response time around 3.07 times. This is the time that only captures an image from the camera. There is more processing time, such as time to process the PSPNet and DQN. Using a setting that takes less time as possible is the best choice.

Figure 4.1

Response Time of Camera for Each Resolution



4.2 Semantic Segmentaion

4.2.1 HRNet-OCR

The HRNet-OCR consists of many layers. When I try to train it with one epoch on the RTX 2070, which has 8 gigabytes of memory, it cannot train due to being out of memory, as shown in Figure 4.2. It tried to allocate 1.39 gigabytes, but the model already used 5.56 gigabytes. It is available only 148.12 megabytes. Thus, it can be assumed that it uses more than 8 gigabytes of memory. This network cannot use due to hardware limitation.

Figure 4.2

GPU's Out of Memory When Running HRNet-OCR

```
RuntimeError: CUDA out of memory. Tried to allocate 1.39 GiB (GPU 0; 8.00 GiB
total capacity; 5.56 GiB already allocated; 148.12 MiB free; 6.18 GiB reserved
in total by PyTorch)
```

4.2.2 Response Time Testing

The response time was tested on my laptop that will be used when testing with the real golf cart.

4.2.2.1 **DeepLabv3.** The average time of the DeepLabv3 at 272x272 is 27.80 milliseconds or 35.97 frames per second, and the maximum time is 59.07 milliseconds or 16.93 frames per second, as shown in Figure 4.3. The average time of DeepLabv3 at 512x512 is 30.06 milliseconds or 33.27 frames per second, and the maximum time is 74.98 milliseconds or 13.34 frames per second, as shown in Figure 4.4.

4.2.2.2 **PSPNet.** The average time of the PSPNet at 272x272 is 9.00 milliseconds or 111.11 frames per second, and the maximum time is 26.84 milliseconds or 37.26 frames per second, as shown in Figure 4.5. The average time of DeepLabv3 at PSPNet is 9.23 milliseconds or 108.34 frames per second, and the maximum time is 48.19 milliseconds or 20.75 frames per second, as shown in Figure 4.6.

The PSPNet was chosen for this thesis. At the resolution of 272x272, the DeepLabv3 took the average response time more than the PSPNet 4.00 times, and the DeepLabv3 took the response time more than the PSPNet 2.20 times in the worst case. At the resolution of 512x512, the DeepLabv3 took the average response time more than the

PSPNet 3.26 times, and the DeepLabv3 took the response time more than the PSPNet 1.56 times in the worst case. From Table 2.2, the DeepLabv3 achieved a mIoU score of 0.813 on the Cityscapes test dataset, and the PSPNet achieved a mIoU score of 0.784. It does not differ much in terms of the mIoU score and quality. Using the PSPNet that takes less response time is the best choice.

Figure 4.3

Response Time of DeepLabv3 at 272x272

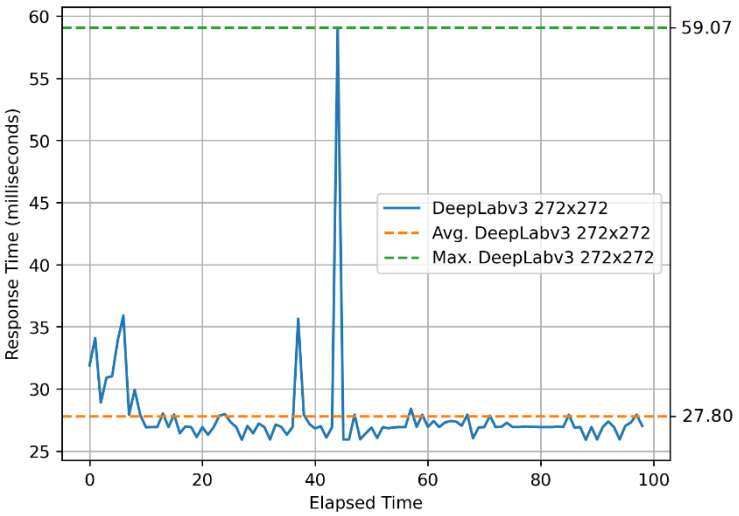


Figure 4.4

Response Time of DeepLabv3 at 512x512

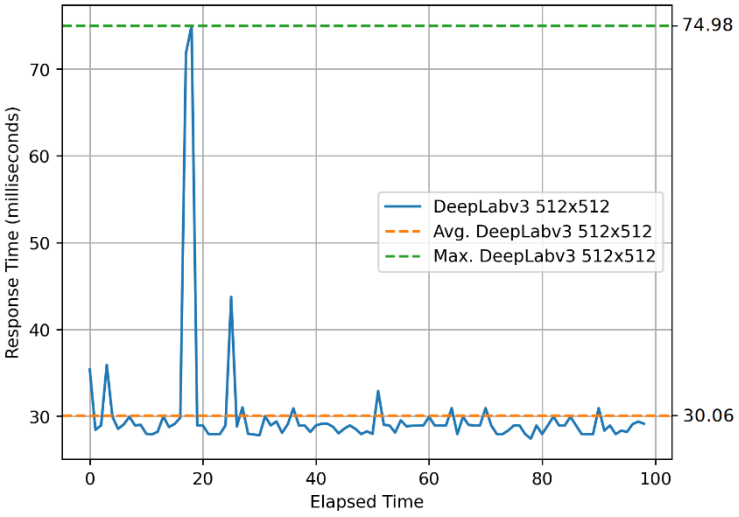


Figure 4.5

Response Time of PSPNet at 272x272

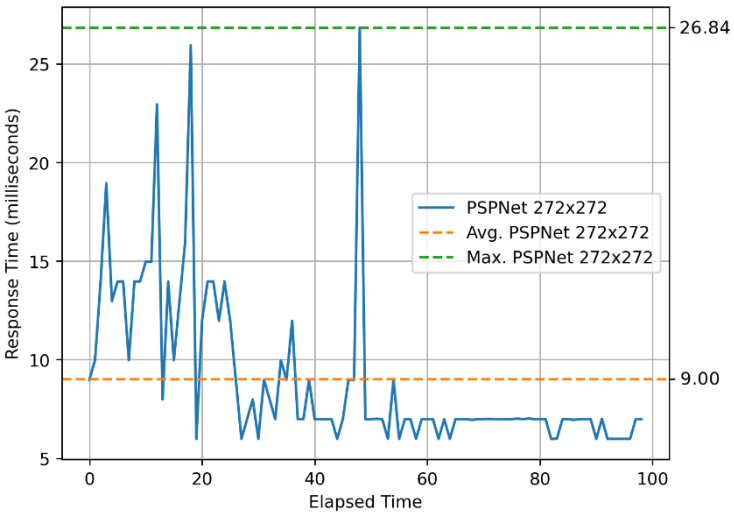
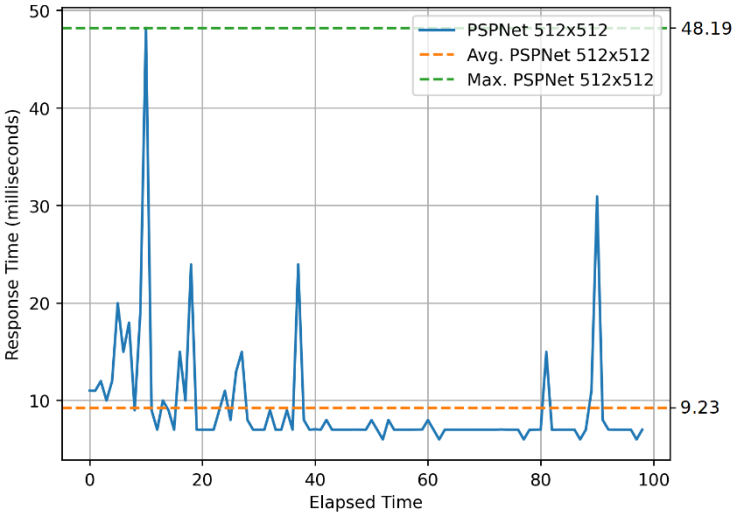


Figure 4.6

Response Time of PSPNet at 512x512



4.2.3 Preprocessing

The mean and standard values that were found from the AIT dataset in each color channel are shown in Table 4.1.

Table 4.1

Mean and Standard Values for Normalization

	Red	Green	Blue
Mean	105.6152	115.1326	118.1004
Standard	68.2132	70.9628	78.0216

4.2.4 Training

***4.2.4.1* PSPNet Experiment 1.** The result of experiment 1 from Section 3.3.4.1 is shown in Figure 4.7. It was trained for 10354 epochs, which took 55.24 hours. The best mIoU is 0.6824 at epoch 5120. The training was stopped because the mIoU score stopped increasing. Examples of the results of the validation set are shown in Figure 4.8. The result is not good, especially the lane markings. The lane markings are hard to detect in the shadows and the far-view.

Figure 4.7

Validation mIoU of PSPNet Experiment 1

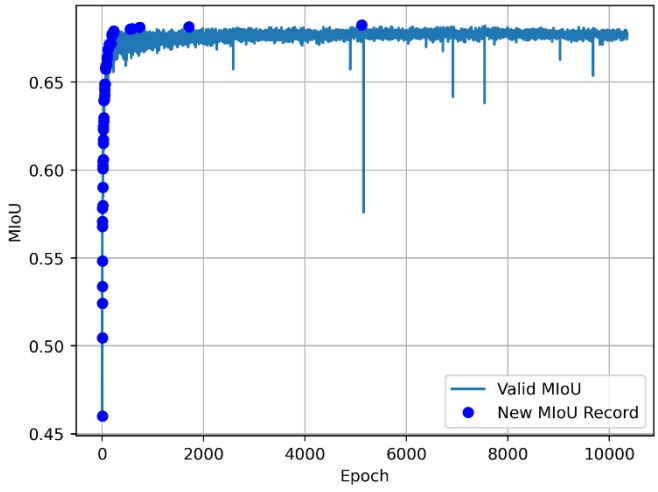
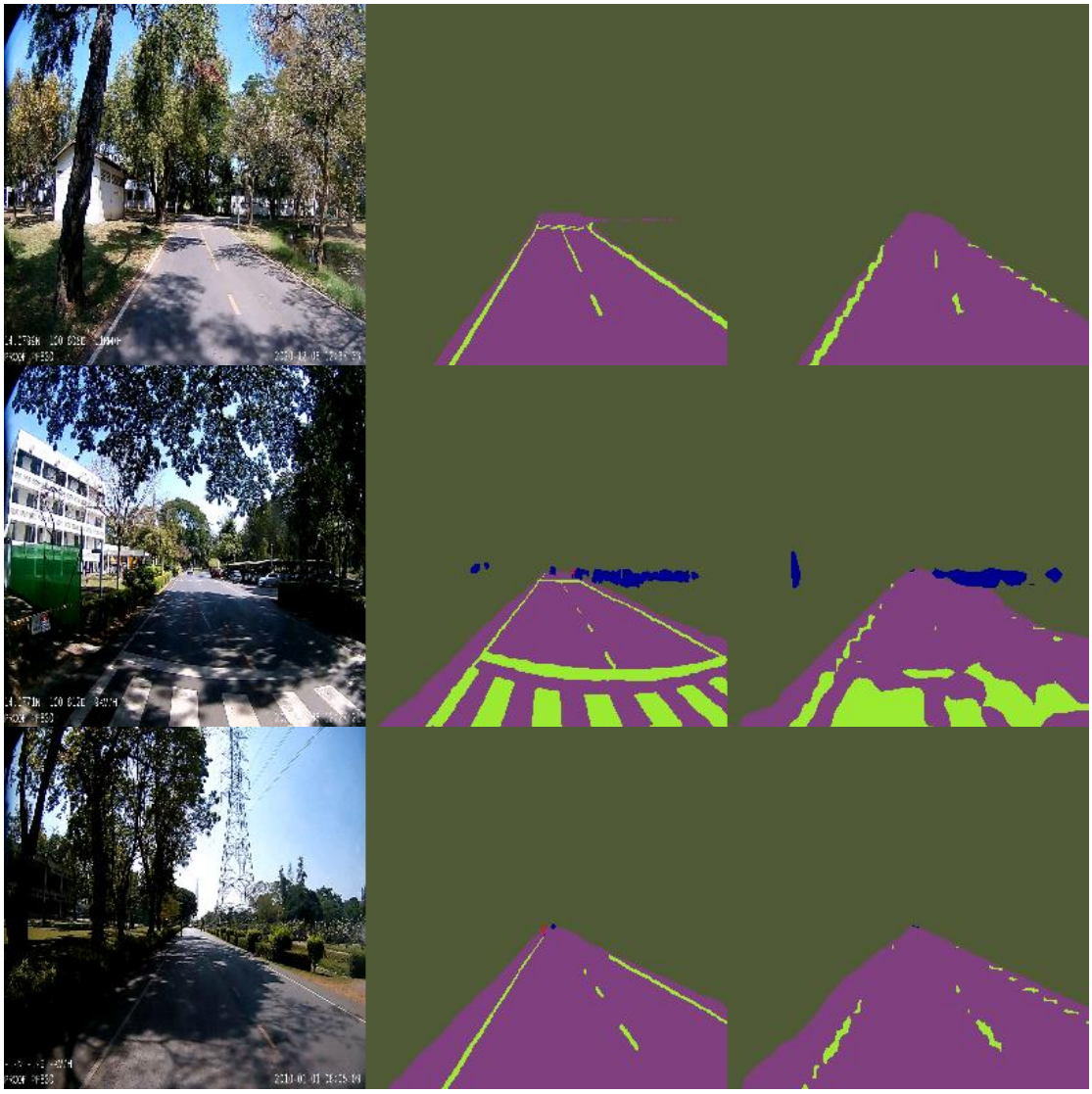


Figure 4.8

Example of the Prediction Result of Validation Set from PSPNet Experiment 1



Note. The left column is input images. The middle column is ground truth images. The last column is prediction images.

4.2.4.2 **PSPNet Experiment 2.** The result of experiment 2 from Section 3.3.4.2 is shown in Figure 4.9. It was trained for 697 epochs, which took 101.41 hours. The best mIoU is 0.7720 at epoch 407. The training was stopped because the mIoU score stopped increasing. Examples of the results of the validation set are shown in Figure 4.11. The result is good even in the shadows and far-view.

Figure 4.9

Validation mIoU of PSPNet Experiment 2

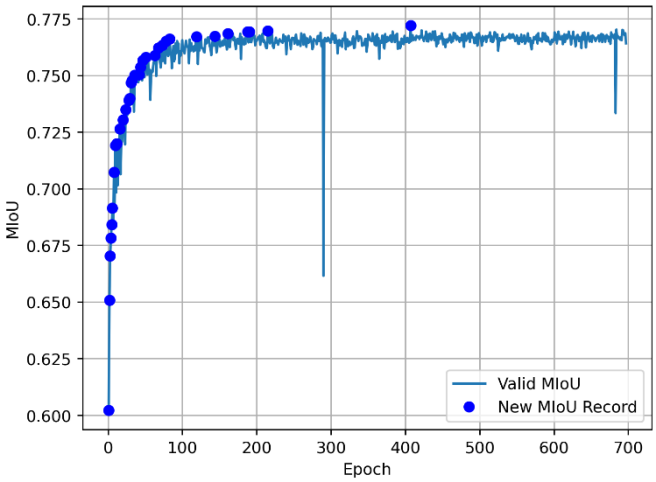
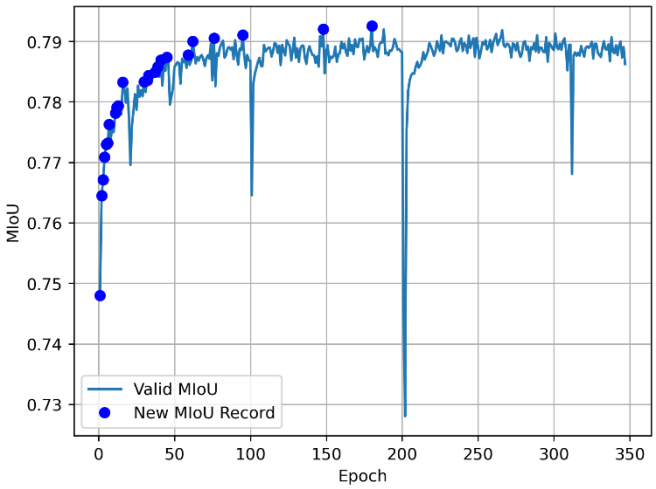


Figure 4.10

Validation mIoU of PSPNet Experiment 3



***4.2.4.3* PSPNet Experiment 3.** The result of experiment 3 from Section 3.3.4.3 is shown in Figure 4.10. It was trained for 347 epochs, which took 6.37 hours. The best mIoU is 0.7925 at epoch 180. The training was stopped because the mIoU score stopped increasing. Examples of the results of the validation set are shown in Figure 4.12. The result is good. It can segment images in the CARLA simulator.

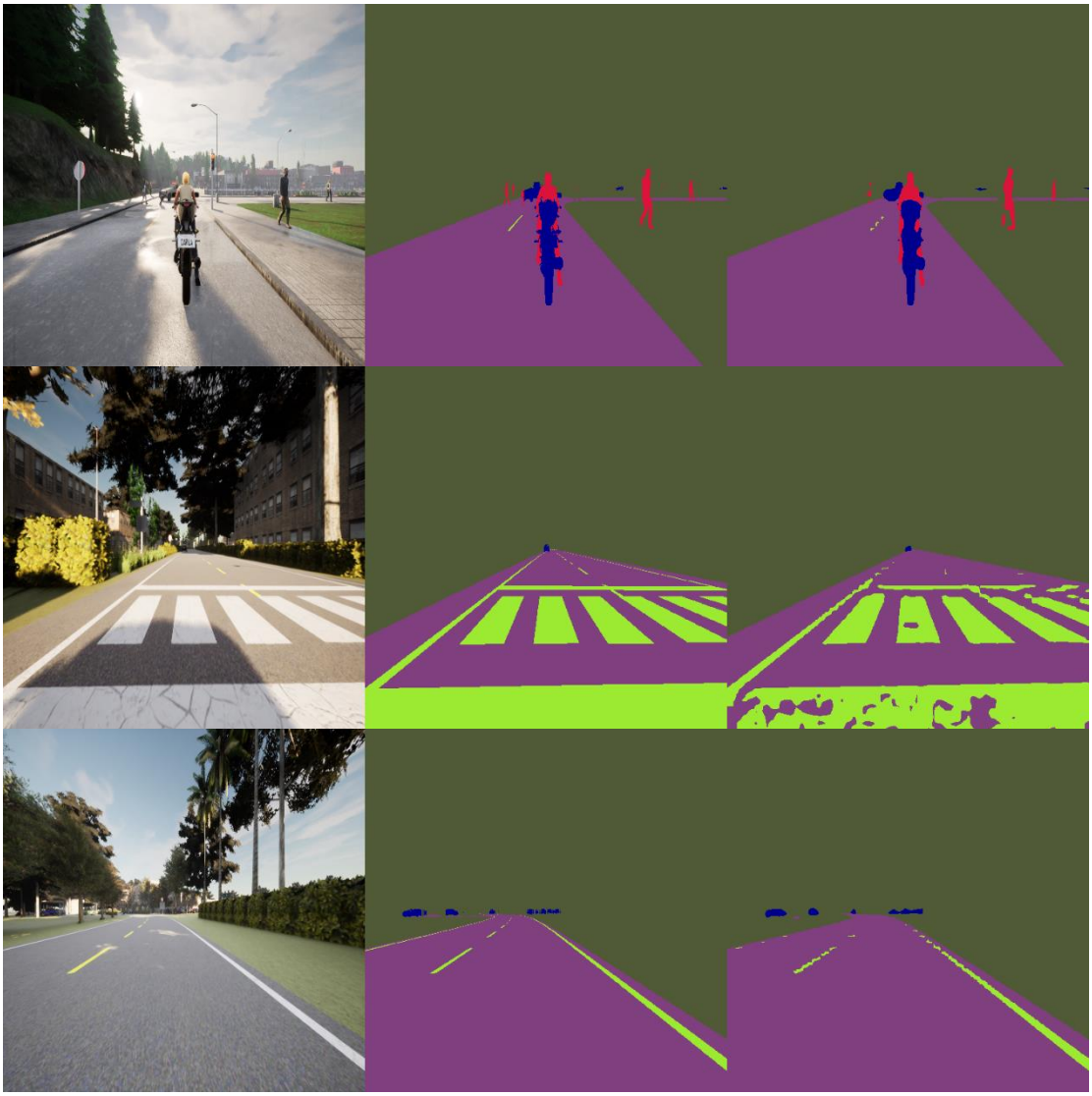
Figure 4.11

Example of the Prediction Result of Validation Set from PSPNet Experiment 2

Note. The left column is input images. The middle column is ground truth images. The last column is prediction images.

Figure 4.12

Example of the Prediction Result of Validation Set from PSPNet Experiment 3



Note. The left column is input images. The middle column is ground truth images. The last column is prediction images

4.3 Double Deep Q-Learning

4.3.1 Training

4.3.1.1 **Double DQN Experiment 1.** The result of experiment 1 from Section 3.5.3.1 is shown in Figure 4.13. It was trained for 172000 steps, which took 86.88 hours. The best mean play score is 0.8231 at step 123300. It can play to the end of 5 scenes at step 14400. This experiment proves that the double DQN algorithm can control a car

using segmented images. The golf cart could drive in the left lane. When it went outside of the left lane, it controlled the steering wheel to come back into the left lane.

Figure 4.13

Mean Play Score of Double DQN Experiment 1

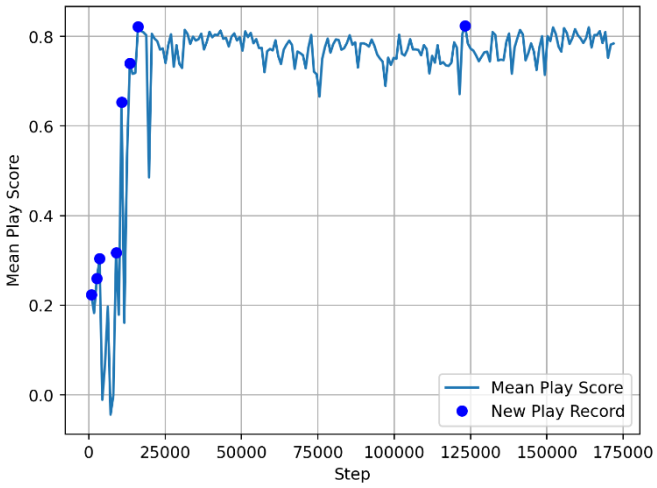
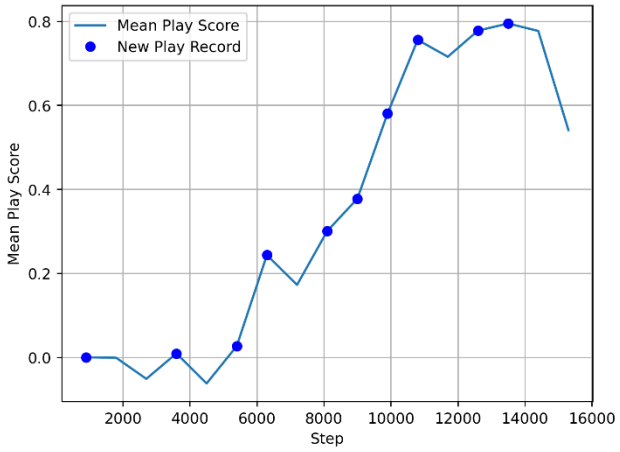


Figure 4.14

Mean Play Score of Double DQN Experiment 2



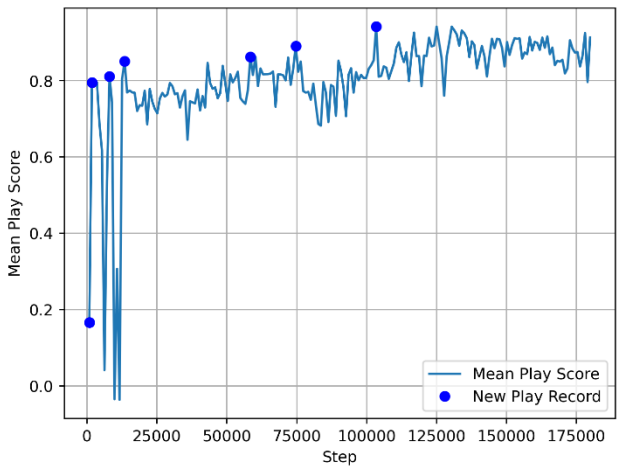
***4.3.1.2* Double DQN Experiment 2.** The result of experiment 2 from Section 3.5.3.2 is shown in Figure 4.14. It was trained for 15300 steps, which took 5.47 hours. The best mean play score is 0.7946 at step 13500. It can play to the end of 5 scenes at

step 10800. The training was stopped because it was concluded that we could use segmented images from the PSPNet instead of perfect segmented images. The PSPNet has more noise than perfect segmented images. Changing the position of the camera and action time have no effect on training the double DQN. The golf cart could drive in the left lane. When it went outside of the left lane, it controlled the steering wheel to come back into the left lane.

***4.3.1.3* Double DQN Experiment 3.** The result of experiment 3 from Section 3.5.3.3 is shown in Figure 4.15. It was trained for 180000 steps, which took 130.96 hours. The best mean play score is 0.9409 at step 103500. It can play to the end of 3 scenes at step 75600. It took more time than previous experiments. The experiments 1 and 2 used 14400 and 10800 steps respectively. This experiment is hard because it has to learn how to drive on left and right curvy roads. The experiments 1 and 2 have only right curvy road. This experiment proves that it can drive on left and right curvy roads, and when doing an action, the action must give a reward that differs from the previous reward. Changing the speed of the golf cart has to change actions. The golf cart could drive in the left lane. It can drive on left and right-curvy roads. When it went outside of the left lane, it controlled the steering wheel to come back into the left lane.

Figure 4.15

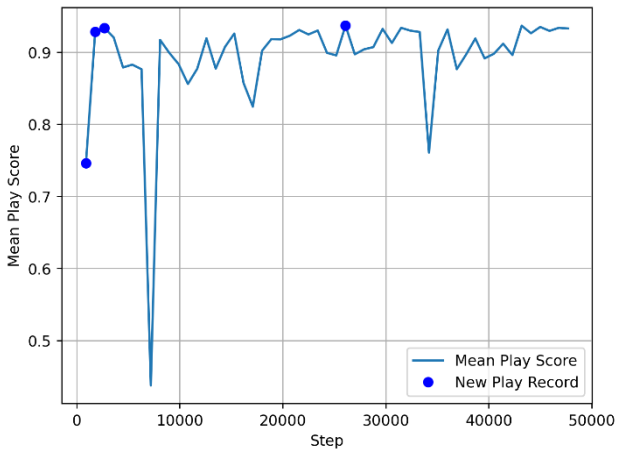
Mean Play Score of Double DQN Experiment 3



***4.3.1.4* Double DQN Experiment 4.** The result of experiment 4 from Section 3.5.3.4 is shown in Figure 4.16. It was trained for 48100 steps, which took 19.06 hours. The best mean play score is 0.9368 at step 26100. It can play to the end of 3 scenes at step 1800, which is very fast because it was reduced the number of actions from 8 actions to 3 actions. The experiment 3 took 75600 steps. Giving a reward in this experiment differs from the previous experiment. The golf cart moved too much on the steering wheel, but it was still in the left lane.

Figure 4.16

Mean Play Score of Double DQN Experiment 4



***4.3.1.5* Double DQN Experiment 5.** The result of experiment 5 from Section 3.5.3.5 is shown in Figure 4.17. It was trained for 94500 steps, which took 58.64 hours. The best mean play score is 0.9196 at step 86400. It can play to the end of 3 scenes at step 24300. Changing the frame time did not affect the result. The golf cart can be driven at 5 frames per second. Giving a reward for this experiment and changing the width of the left lane can improve the results. The golf cart was driven perfectly. It can drive in the middle of the road.

Figure 4.17

Mean Play Score of Double DQN Experiment 5

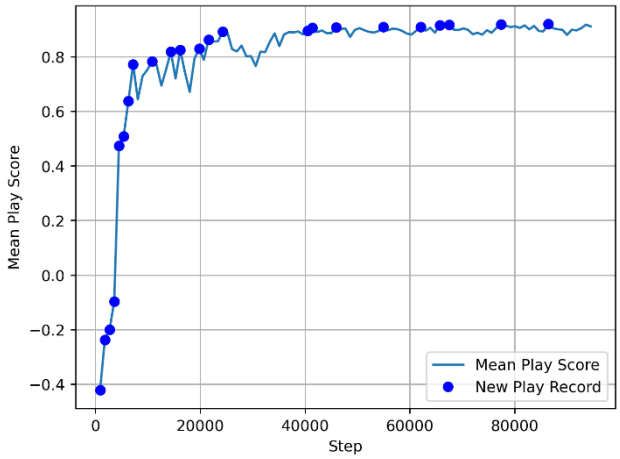
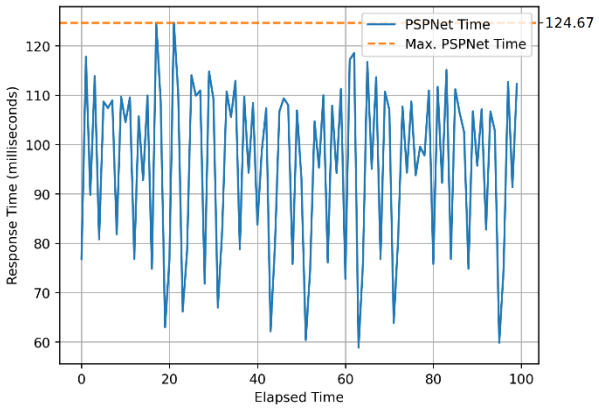


Figure 4.18

Response Time of PSPNet when Real Test



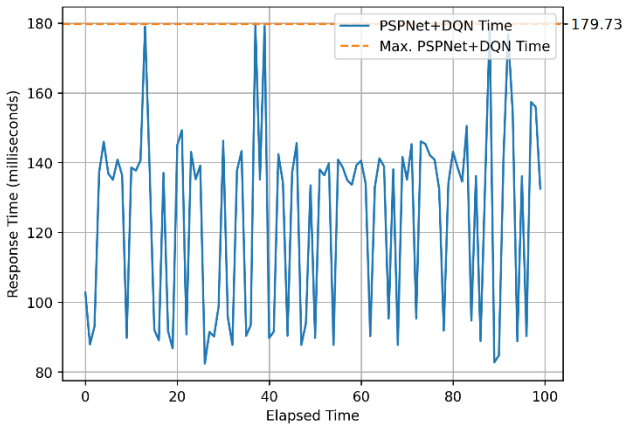
4.3.2 Testing

***4.3.2.1* Testing from Experiment 3.** Testing the real golf cart using the trained model from experiment 3, the golf cart was almost driven successfully. It was controlled well on straight roads, but not well on curvy roads. In the end, it drove off the road. This model takes too long time to make a decision. Figure 4.18 shows response times when processed by the PSPNet. This experiment set the frame time at 0.1 seconds, but when running in the real test, it was run with the DQN every 0.4 seconds. This may be the reason why it processed for more than 0.1 seconds, and from Figure 4.19, almost

all of them processed more than 0.1 seconds. Setting the frame time to 0.2 seconds is the best way to process the networks.

Figure 4.19

Response Time of PSPNet and DQN when Real Test



4.3.2.2 **Testing from Experiment 5.** The golf cart was tested using the trained model from experiment 5. The golf cart was driven perfectly on the road that was trained. It was not just driving in the left lane, but it could drive in the middle of the left lane as well. It can still drive on a road that it has never seen, and it can drive on a road without line markings.

CHAPTER 5
CONCLUSIONS

5.1 Conclusion

The golf cart is controlled using the double DQN algorithm. The input of the double DQN is segmented images which come from a semantic segmentation model. The PSPNet and DeepLabv3 were chosen for testing. At a resolution of 272x272, the PSPNet can run 37.26 frames per second, and the DeepLabv3 can run 5.25 frames per second in the worst case. At a resolution of 512x512, the PSPNet can run at 20.75 frames per second, and the DeepLabv3 can run at 5.97 frames per second in the worst case. The PSPNet was chosen to be used in this thesis because it can provide better times on both resolutions. The PSPNet was trained using the AIT dataset and the Mapillary Vistas dataset, which is a public dataset. The best mIoU of the PSPNet training at the resolution of 272x272 is 0.6824, and the best mIoU of the PSPNet training at the resolution of 512x512 is 0.7925. The resolution of 512x512 was chosen because it provides a better score and quality of segmented images. When the PSPNet was applied at the same time as the DQN, it took more than 0.1 seconds to process. Thus, the frame time was set at 0.2 seconds or 5 frames per second. An action will be predicted and sent every 0.4 seconds. From a lot of trial and error of training the double DQN, reducing the reward mask of the left lane can provide a driving in the middle of the left lane. The reward will be deducted by 0.2 when moving the steering wheel to make it move the steering wheel as less as possible. Giving the reward -1 when it goes outside the left lane to make it more afraid to move to the outside.

To summarize, this thesis finds the best semantic segmentation model to segment images, and finds the best way to give a reward to the double DQN to control the golf cart in the left lane. The hardest part about using the double DQN is designing a reward function.

5.2 Recommendations

1. Train the semantic segmentation network with more data from the public dataset, and augment the dataset by adding low brightness and high brightness to make the network run on every road and light condition.

2. Train the double DQN with more scenes to make the car can be driven on every road.
3. Use an encoder network, such as ResNet, instead of my designed network in the double DQN part, and try to adjust the hyperparameters. It may improve the speed of training.

REFERENCES

Arzt, S. (2016, October 23). *Deep Learning Cars* [Video]. YouTube. https://youtu.be/Aut32pR5PQA

Chishti, S. O. A., Riaz, S., Zaib, M. B., & Nauman, M. (2018, November 1-2). *Self-Driving Cars Using CNN and Q-Learning* [Conference session]. 2018 IEEE 21st International Multi-Topic Conference (INMIC), Karachi, Pakistan. https://doi.org/10.1109/INMIC.2018.8595684

Choudhary, A. (2019, April 18). A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017, November 13-15). *CARLA: An Open Urban Driving Simulator* [Conference session]. the 1st Conference on Robot Learning (CoRL), Mountain View, California, United States of America. https://arxiv.org/abs/1711.03938

Fayjie, A. R., Hossain, S., Oualid, D., & Lee, D. J. (2018, June 26-30). *Driverless Car: Autonomous Driving Using Deep Reinforcement Learning in Urban Environment* [Conference session]. 2018 15th International Conference on Ubiquitous Robots (UR), Honolulu, Hawaii, United States. https://doi.org/10.1109/URAI.2018.8441797

Gulli, A., Kapoor, A., & Pal, S. (2020, January 9). Using the CNN Architecture in Image Processing. *Open Data Science*. https://opendatascience.com/using-the-cnn-architecture-in-image-processing

Heidenreich, H. (2018, December 5). What are the types of machine learning? *Towards Data Science*. https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f

Kinsley, H. (2017, April 10). *Python Plays: Grand Theft Auto V* [Video]. YouTube. https://www.youtube.com/playlist?list=PLQVvvaa0QuDeETZEOy4VdocT7TOjfSA8a

Kinsley, H. (2019, September 16). *Self-driving cars with Carla and Python* [Video]. YouTube. https://www.youtube.com/playlist?list=PLQVvvaa0QuDeI12McNQdnTlWz9XlCa0uo

Li, Y., Guo, L., Rao, J., Xu, L., & Jin, S. (2018). Road Segmentation Based on Hybrid Convolutional Network for High-Resolution Visible Remote Sensing Image. *IEEE Geoscience and Remote Sensing Letters*, 16(4), 613-617. https://doi.org/10.1109/LGRS.2018.2878771

Lim, K. L., Drage, T., & Bräunl, T. (2017, November 16-18). *Implementation of semantic segmentation for road and lane detection on an autonomous ground vehicle with LIDAR* [Conference session]. 2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), Daegu, South Korea. https://doi.org/10.1109/MFI.2017.8170358

Liu, Y. H. (2019). *PyTorch 1.x Reinforcement Learning Cookbook*. Packt.

Neuhold, G., Ollmann, T., Bulò, S. R., & Kontschieder, P. (2017, October 22-29). *The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes* [Conference session]. 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy. https://doi.org/10.1109/ICCV.2017.534

Okuyama, T., Okuyama, T., & Upadhay, J. (2018, March 1-3). *Autonomous Driving System based on Deep Q Learnig* [Conference session]. 2018 International Conference on Intelligent Autonomous Systems (ICoIAS), Singapore, Singapore. https://doi.org/10.1109/ICoIAS.2018.8494053

Stojnic, R., Taylor, R., Kardas, M., Kerkez, V., Viaud, L., Saravia, E., & Cucurull, G. (2021, May 31). Semantic Segmentation. *Papers with Code*. https://paperswithcode.com/task/semantic-segmentation

Tiu, E. (2019, August 10). Metrics to Evaluate your Semantic Segmentation Model. *Towards Data Science*. https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2

Yakubovskiy, P. (2019, April 20). segmentation_models.pytorch. *GitHub*. https://github.com/qubvel/segmentation_models.pytorch

Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2017, July 21-26). *Pyramid Scene Parsing Network* [Conference session]. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, United States. https://doi.org/10.1109/CVPR.2017.660