Robotics – Final Project Report

Student: Han Le - 111679478

Project title: Autonomous car simulation using Convolutional Neural Network (CNN).

1. Introduction

The aim of this project is to implement a pipeline for training and simulating autonomous car in 3D simulation environment using CNN including collecting training/test data, designing a CNN network to train and run autonomous car in the simulation environment which is a Unity based game platform provided by Udacity [2]. The Simulator is a simple car driving game where there is only the car itself running on a closed loop road, there is no traffic law interfering the game. It allows users to collect data by manually steering the car' angles to move the car along the road. The data consisting of road views, steering angles, and speed will be used to train a CNN network and then use this model to drive the car autonomously around the track. The project originally provides a framework using Keras [2, 3] to train, validate and test the model but Pytorch will be used for this project.

2 Data processing

2.1 Data collection

There is no data provided, it must be collected manually by driving the car (playing the game) along the road. During playing, the simulator will capture 3 views representing left, center, and right cameras as well as the steering angle and the car's velocity. Those 3 views are saved as .jpg images and numerical data is saved to a .csv file along with the file names of corresponding images of the scene. It takes around 5 minutes to complete collecting data for 1 lap that generates around 4000 images. There are 2 tracks and 6 graphic quality settings but for simplicity, 3 graphics options (Fastest, Simple, and Beautiful) chosen for data collection. Hence, there are 6 combinations which will be collected for 5 laps each. Therefore, the total images are around 6*5*4000 = **120000** images and total time is around 6*5*5 = 150 mins = **2.5hrs**

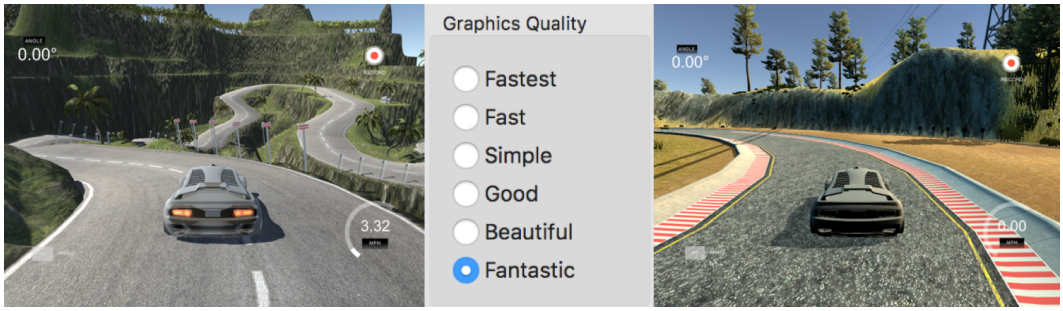

Figure 1: Simulator's user interface

2.2 Data augmentation

As a normal practice in training a convolutional neural network, data augmentation is often used to improve the learning ability of the model and to compensate the lack of training data. In this project, several common augmentation techniques are used such as random flip, random translation, brightness adjustment, and crop un-useful areas.

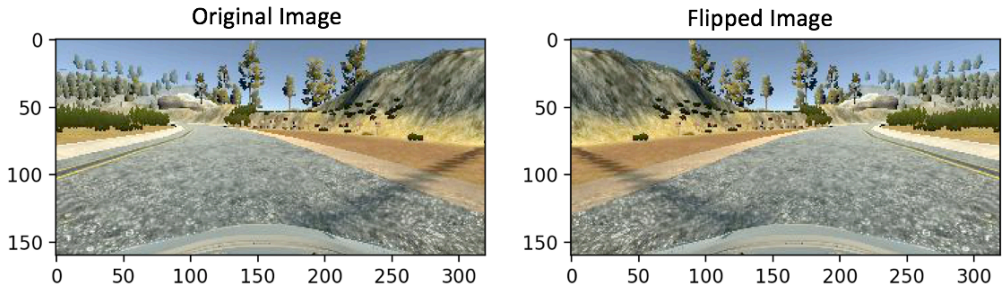

Figure 2: Original image and flipped one

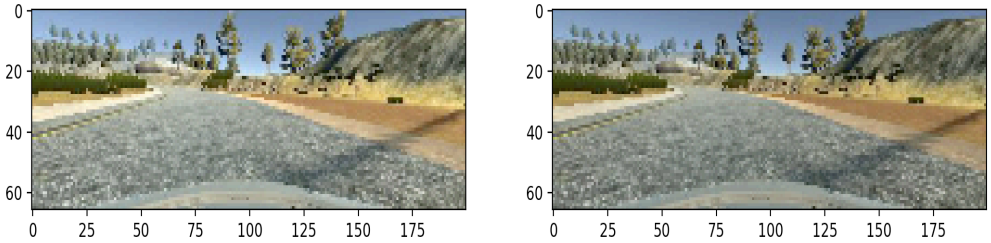

Figure 3: Original image and the one with brightness randomly adjusted

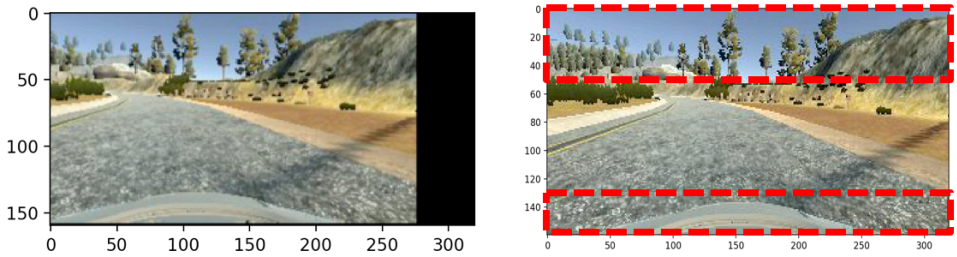

Figure 4: Translated and cropped image

3 Model implementation

A simple CNN model including two Convolutional layers, 2 MaxPooling, and 4 fully connected layers is used first to optimize the data augmentation techniques and to choose appropriate optimizer and hyper-parameters for the problem. The simple model quickly works well for when sufficient data augmentation included but it only works for a certain number of testing cases. As a result, a more complex model is required.

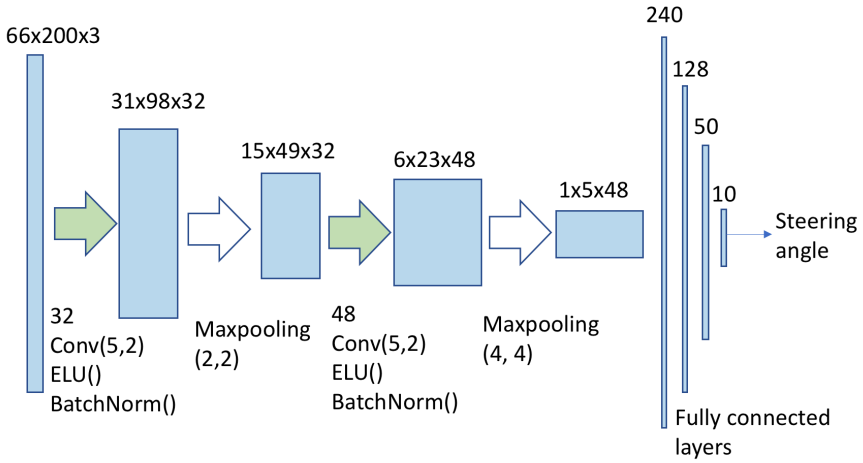

Figure 5: A simple CNN model

As suggested by Nvidia [1], a model with below architecture is implemented. It includes 3 successive 5x5 kernel convolutional layers with stride of 2, followed by 2 convolutional layers with kernel of 3 and stride of 1. There are 4 fully connected layers at the top of the model. It is suggested that this architecture works in practice but it did not work well in this project. As a result, another model which is inspired by Nvidia's design is proposed named "final model".

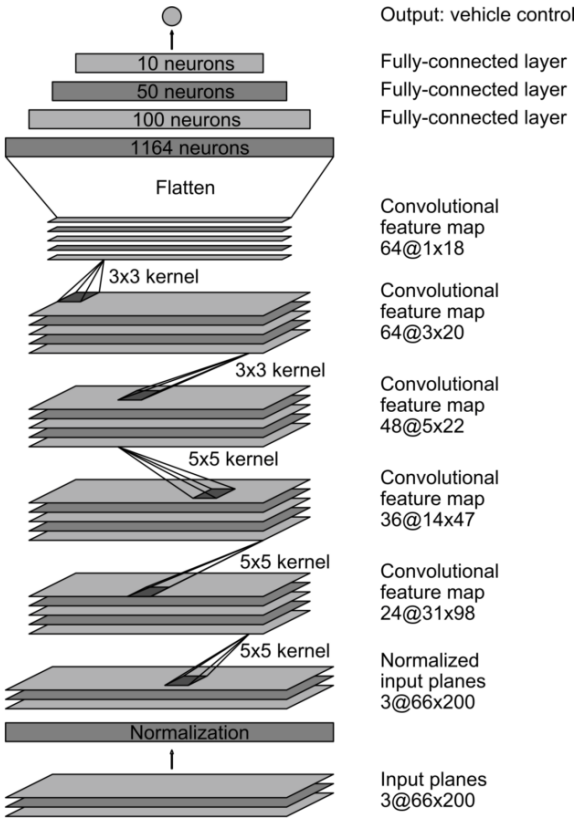

Figure 6: Nvidia's suggested model

The key idea of the above model is that convolutional layer with large kernel of 5 are used near the input layer to capture more global information of the input. As a result, the model will be more stable with different graphic settings of the input images.

Inspired by Nvidia's design, the proposed model also uses convolutional layer with kernel size of 5 for the first 2 layers. However, the MaxPooling is used in the next layer instead of the third convolutional layer. In addition, BatchNorm, ReLU, and Dropout show a significant improvement to the performance of the model.

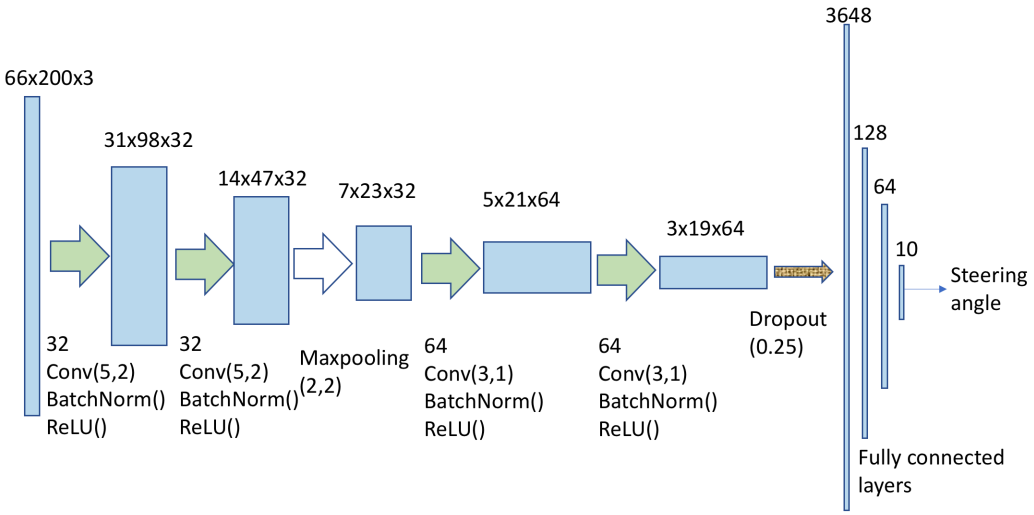

Figure 7: Final model

4. Training

During training phase, the input images which are the view from left, center, or right camera are fed to the CNN after going through augmentation module. The steering angles collected during data collection are used as supervised labels to optimize the model. Mean squared error is used as the loss function since the output and label are scalars.

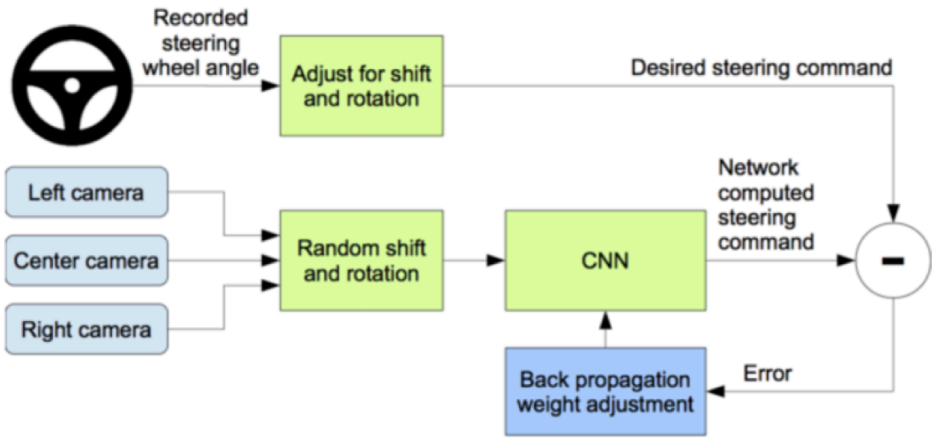

Figure 8: Training phase

As normal, the data is divided into training and validation set. The model is trained on training set and evaluated on validation set. Models with the lowest value of MSE on validation set are saved as final models and used to test on the road with different graphic settings. Based on characteristic of the road views and training process, road 1 is much easier to train than road 2.

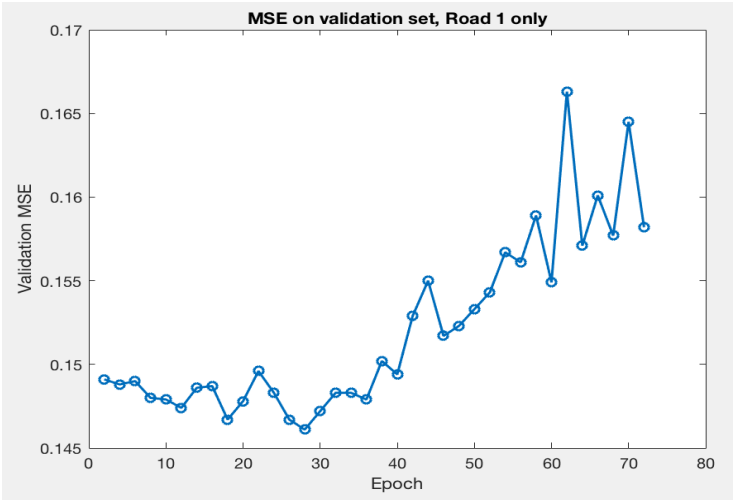

Figure 9.1: MSE of validation set when trained on Road 1 data

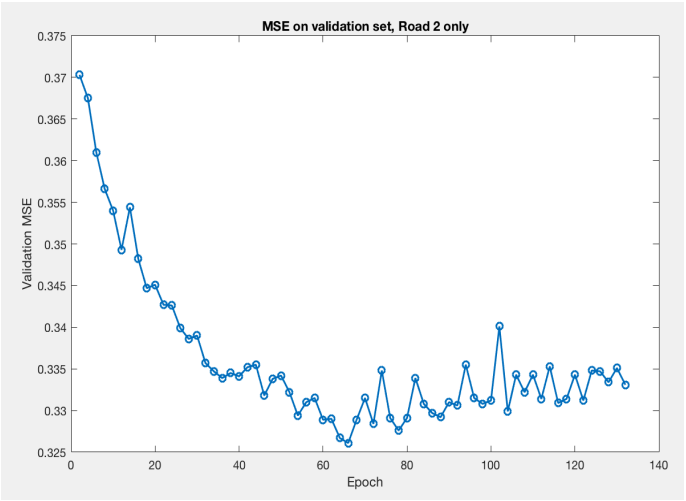

Figure 9.1: MSE of validation set when trained on Road 2 data

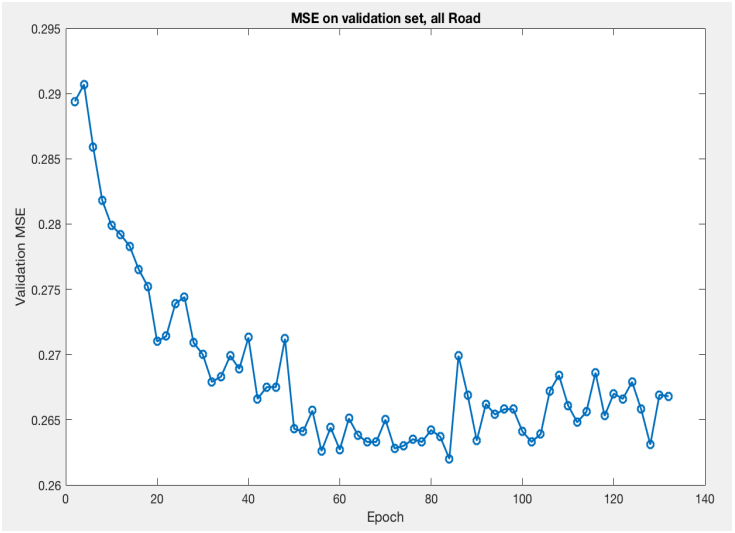

Figure 9.1: MSE of validation set when trained on both Road 1 and Road 2 data

5. Testing

During testing phase, the image from the center camera is fed into the trained model, the output of the network which is the steering angle is then used to compute the throttle that sent to control the movement of the car in autonomous mode.

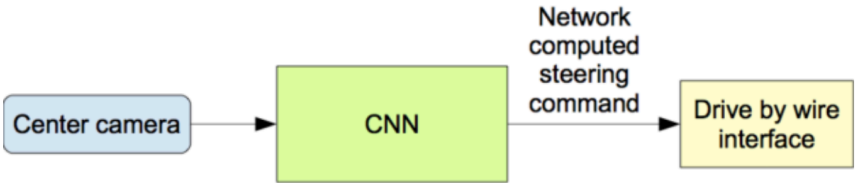

Figure 10: Testing phase

Three set of training data including road 1 only, road 2 only, and mixed both road 1 and road 2 images are used to train the same model architecture resulting in 3 trained models which are used in testing phase against different road types and graphic settings. The table below summarizes the result of all test cases.

Training data	**Test on Road 1**	**Test on Road 2**
Road 1 only	 Youtube Link [4]	
Road 2 only		 Youtube Link [5]
Both road 1 and road 2	 Youtube Link [6]	

It is not surprised that the model which is trained only on 1 type of the road can perform well on the same road type but performs poorly on the other road. The model which is trained on both roads performs well on the Road 1 but slightly worse on Road 2. This is expected because the Road 2 is much complicated than the Road 1.

6. Conclusion

In this project, a pipeline for training and simulating autonomous car in 3D simulation environment using CNN is implemented and analyzed. It has shown that the CNN model is capable of learning useful visual feature not only in a same distribution (same road type) but also in different data distribution (different road views). As the result, it is able to successfully generate appropriate control signals to drive the car in different environment settings.

REFERENCES

[1] Mariusz Bojarski, Ben Firner, Beat Flepp, Larry Jackel, Urs Muller, Karol Zieba and Davide Del Testa. End-to-End Deep Learning for Self-Driving Cars. August 17, 2016.
https://devblogs.nvidia.com/deep-learning-self-driving-cars/ [Online accessed 15-May-2018]
[2] https://github.com/udacity/self-driving-car-sim
[3] https://github.com/llSourcell/How_to_simulate_a_self_driving_car
[4] https://www.youtube.com/watch?v=nTf0mYLwEAE
[5] https://www.youtube.com/watch?v=yFSCDUysG_E&t=1s
[6] https://www.youtube.com/watch?v=ghRPx6zP2I4&t=35s