Self-Driving Car Simulation using Deep Learning

Ronak Sharma
University at Buffalo
ronaksha@buffalo.edu

Nikhil Arora
University at Buffalo
nikhilar@buffalo.edu

Abstract

In this project we will be using Udacity Self Driving Car simulator which is an open source simulator software for Researchers and developers. In the project we have designed two different tracks, one for Training and other for Testing. First, we obtain Training data from the simulator as images of what the car would see while it drives with manual control and later we feed the Training images to the Deep learning Model for it to learn through them. The Deep learning model deployed is taking the images as input and the steering angle as target variable. To comprehend, we are going to use camera data as model input and expect it to predict the steering angle in the [-1,1] range. Through this project we will be exhibiting the performance of multiple deep learning models in a real-world scenario. The back bone of this project is supported by Nvidia's end-to-end self-driving car paper [1].

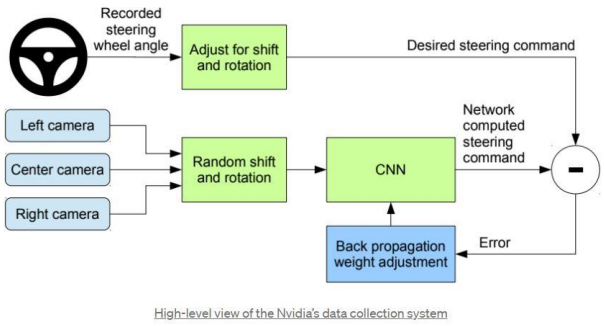

Figure 1: High Level View [2]

1. Introduction

CNN architectures give a different approach on looking at image data. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images. Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations [3].

While CNNs have been in used in many different applications and from a long time but in recent years CNNs have got popularity. This is because of the development in computational power of Processors and the large scale publicly available organized labelled date, which makes it easier for the neural network to learn on the data and also allows them the necessary computational freedom.

In 2016 Nvidia published a research based on the ability of CNNs to automatically read the features from input images. It aims to extend the functionality of an earlier model called DAVE which is another neural network model developed by DARPA. DAVE was trained on data from various off-road terrain and then trained on similar environment but it couldn't do well while testing. Nvidia's research aimed at creating another model which would not require any human designed features such as lane markings, signals, road maps, or traffic regulations. It would learn the features automatically using a left camera view and a right camera view.

In this project we aimed at getting higher accuracy than the Nvidia model and comparing various convolutional neural network architectures with the baseline evaluation results. We used Udacity self-driving car simulator which is a free to use software, it gives freedom on creating our own training and testing tracks, it also comes with a few of them pre-created. In the simulator, we have front right, front left and a center camera which captures image frames from the simulator and saves them on an external location. Before training the model, the input images were preprocessed in order to eliminate training outliers. In this project we implemented and compared VGG16 and Resnet Architectures with the Baseline Nvidia model. To sum it up, we were able to achieve a higher test accuracy in this project which was advocated by simulating the test drive all by itself.

2. Task Statement

Developing and Training of a deep learning model that can drive a car by itself. It is a supervised regression problem that can train on the road images from the car camera and steering angles.

The Udacity driving simulator saves frames from three different front facing cameras. Along with frames the simulator also saves various parameters like throttle, steering angle, and speed. The task statement for this project aligns along with Collecting the data from training tracks, data preprocessing, developing the deep learning model, training the model with the data collected, and Testing the trained model on test tracks.

Tasks statement for this project are: -

1. Installation of Simulation Environment.
2. Collecting of Data from Training track.
3. Development of end-to-end deep learning model.
4. Training and Testing
5. Evaluation.

2.1. Installation of Simulator

To use the Udacity simulator, we need Unity game making engine. For downloading the unity simulator, we used Git LFS to clone the official Udacity self-driving car sim repository [4]. Next step is to load the scenes, here we opened the pre-made lake scene for training and jungle track for the testing of our model. The simulator looks like as shown in the picture below:

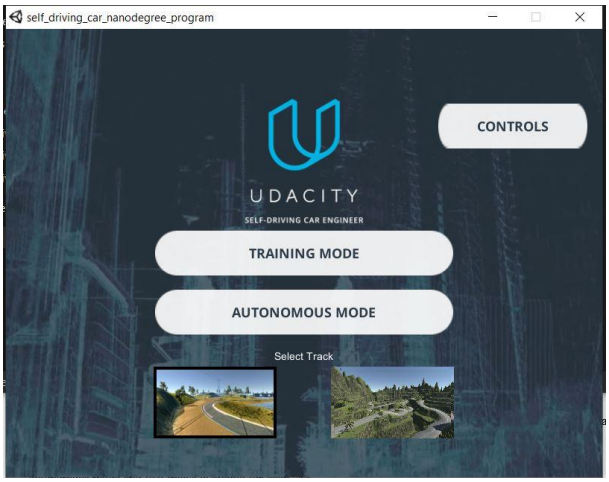

Figure 2 Udacity Simulator for Training and Testing

2.2. Data Collection

There are two modes in the simulator, training mode and autonomous mode and two tracks available for each. As stated above, we used lake scene for the training purpose. The car has three cameras installed, “left, right and center”, also, give four parameters namely, “steering, throttle, brake and speed”. Ideally, to get better data set, one should drive as many laps one can, we drove for 4 laps. When you click record on the simulator the simulator will rewind whole trip collecting data and storing it in two folders namely “driving log” which is csv file and “IMG” folder containing all the images.

2.3. Deep Learning Model Development

In this Project, we have used pre-trained Neural networks and self-designed Neural Networks. Models used in this project are:

Table 1: Models Used

VGG16	Self-Designed
Resnet50	Pre-Trained
Modified + Nvidia model	Self-Designed
InceptionV2	Pre-Trained

2.4. Training and Testing

As explained in Data collection, front facing camera feed from the simulator was used to collect image data and image labels as Images and labels in csv file separately.
Data Collected was preprocessed and split into Training and Testing data. Training data was fed into the Deep Neural networks and weight files were trained which was saved for testing.
The final loss of all the models were compared based on it, best performing model was selected. Selected model was tested on the simulator using another python script created in Anaconda. Anaconda provides an efficient medium for pushing the trained model in the simulator. Best Performing model was ran into the simulator in autonomous mode, where it faced real world environment and performed well on unseen tracks that were totally different from the tracks it was trained on.

2.5. Evaluation

For Evaluation of the Model, comparisons were made on their final Training loss. Mean Squared Error function was used as loss function in this project. Mean Squared Error is measured as the average of squared difference between the predictions and actual observations [5].
MSE is also very useful in regression problems as it measures the distance of the prediction with the actual observation, thus, penalizing the points that are far away from the regression line.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Mean Squared Error

Figure 3: Mean Squared Error [6]

Based on the Loss calculated by the model, weights are trained during Training. Mean squared error is such a common and great loss function; **it is grounded in extremely reasonable probabilistic assumptions**: that given the value of some independent variable(s) (a.k.a. some state of the world), the dependent variable(s) are distributed according to a Gaussian distribution and this fits well for this Regression project [7].

3. Method

Data collection, Pre-processing and Development Deep Neural Network, Training and Testing are the main methods of this project.

3.1. Data Pre-Processing

Image data from three camera angles are stacked together and are fed for pre-processing. Pre-processing is mainly cropping the image. Images are cropped to prevent the model from learning on outliers and data other than the tracks. Image data then were converted to YUB color encoding which is mentioned in the Nvidia model.

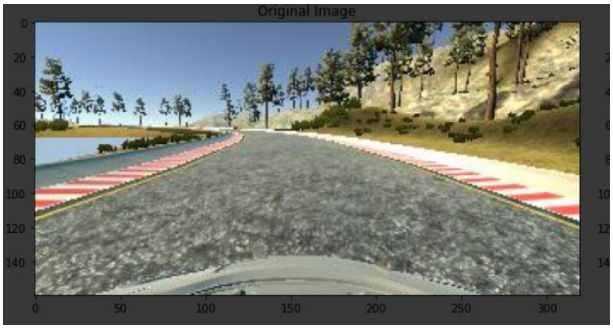

Figure 4: Image Before the Pre-Processing

Processed data is then fed into the Deep Neural Network for Training.

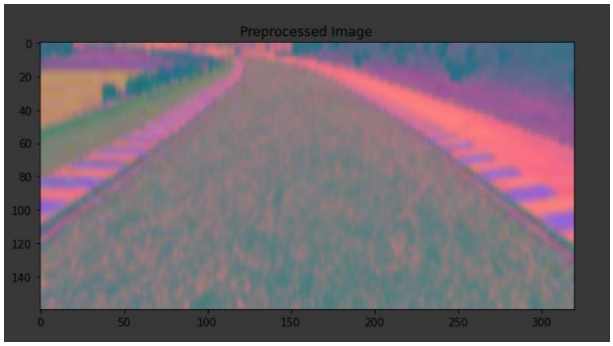

Figure 5: Image after Pre-Processing

3.2. Deep Learning Models

Models used in this project are:

- VGG16
- Resnet50
- InceptionV2
- Modified + Nvidia Model

For the pre-trained models, Resnet50 and Inception V2 were downloaded directly from keras library and because of the reason that they contain large number of layers, we froze all the layers from training except the last 4. This enabled to make a model which is compatible for our dataset.

Model: "sequential"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 4, 4, 2048)	23587712
dropout (Dropout)	(None, 4, 4, 2048)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 100)	3276900
dropout_1 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 50)	5050
dropout_2 (Dropout)	(None, 50)	0
dense_2 (Dense)	(None, 10)	510
dropout_3 (Dropout)	(None, 10)	0
dense_3 (Dense)	(None, 1)	11

Total params: 26,870,183
Trainable params: 4,337,191
Non-trainable params: 22,532,992

Figure 6 Model Summary of ResNet50

For VGG16 and modified model, we created a model with 5 layers each and 4 layers of NVIDIA model, all fully connected shown in figure 7. All the models are compiled using Adam optimizer with its default learning rate. Also, we tried to use SGD optimizer but SGD requires a lot of epochs to train a model accurately so we

decided to go with Adam optimizer. In the end, for training the model, we used checkpoints and take into point the training loss.

Layer (type)	Output Shape	Param #
lambda (Lambda)	(None, 160, 320, 3)	0
cropping2d (Cropping2D)	(None, 65, 320, 3)	0
conv2d (Conv2D)	(None, 31, 158, 24)	1824
dropout (Dropout)	(None, 31, 158, 24)	0
conv2d_1 (Conv2D)	(None, 14, 77, 36)	21636
dropout_1 (Dropout)	(None, 14, 77, 36)	0
conv2d_2 (Conv2D)	(None, 5, 37, 48)	43248
dropout_2 (Dropout)	(None, 5, 37, 48)	0
conv2d_3 (Conv2D)	(None, 3, 35, 64)	27712
dropout_3 (Dropout)	(None, 3, 35, 64)	0
conv2d_4 (Conv2D)	(None, 1, 33, 64)	36928
flatten (Flatten)	(None, 2112)	0
dense (Dense)	(None, 100)	211300
dense_1 (Dense)	(None, 50)	5050
dense_2 (Dense)	(None, 10)	510
dense_3 (Dense)	(None, 1)	11

Total params: 348,219
Trainable params: 348,219
Non-trainable params: 0

Figure 7 Model Summary of submitted model

After training, we chose the model in which loss is reduced to the least number and that is our modified model which reduced from 9% to 3% and saved the best model in h5 file.

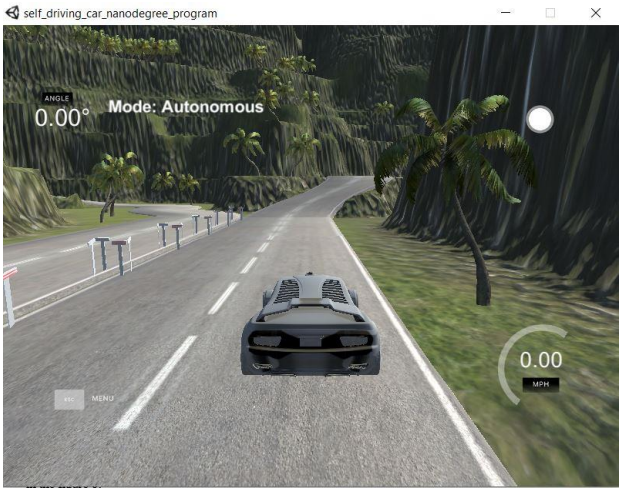

Figure 8 Jungle track for Autonomous testing

For testing the model, we used different track on the simulator, "jungle track". The trained model has never seen this track before, so this test gives the best and raw results on how good the model performs in new environments. The testing script uses different python utilities like TensorFlow, SciPy, eventlet, etc. to load the run the saved trained model and run it real-time on the Udacity simulator. While running on the simulator for the autonomous version this time, Anaconda prompt displays steering angle and speed of the car in real-time as shown in the figure 8:

4. Results

Following table shows the comparison between loss of all the models we compared:

Model	Loss
Modified	0.0319
VGG16	0.0961
Resnet50	0.0932
InceptionV2	0.1035

Table 2 Model Comparison

As we can see from the table 2 that modified model has the lowest loss, we saved this model and used it for testing it on new track.
We have included a video in the supplementary folder as reference to how our model performs in an environment it has not seen before.

5. Future Work

The project can be extended by using real-world data with two different sets, one having lane markings and other consisting of image data. Performance of the model can be increase by fine tuning it more by collecting large dataset and different optimizers.

References

[1] https://towardsdatascience.com/how-to-train-your-self-driving-car-using-deep-learning-ce8ff76119cb
[2] https://towardsdatascience.com/how-to-train-your-self-driving-car-using-deep-learning-ce8ff76119cb
[3] https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf
[4] https://github.com/udacity/self-driving-car-sim
[5] https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23
[6] https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23
[7] https://towardsdatascience.com/where-does-mean-squared-error-mse-come-from-2002bbbd7806
[8] https://github.com/RamAIbot/Self_Driving_car_Behavioral_Cloning