

C. D. Vijay
Udacity Car ND Program

Report P3: Behavioral Cloning

Table of Contents

1. Model Architecture	2
2. Training Strategy	3
3. Track and Simulation Images	4
4. Files Submitted in the Project	7
5. Reference	7

Model Architecture

1. NVIDIA architecture was employed with several modifications

A modified NVIDIA architecture [1] was employed and tailored to meet more modest memory and computation resources available.

This network contains five convolution layers [Convolution2D()] with a kernel size of 5x5 and 3x3 with depth ranging from 24-64 (p34nvmod52.py, lines 114-126), interspersed with the same number of MaxPooling layers [MaxPooling2D()]. Each convolutional layer has an activation component (RELU). The network is flattened followed by three [Fully Connected] Dense () layers (ibid., lines 130-134).

Data is normalized in the model using a Keras lambda layer (ibid., line 105) and cropped to remove the extraneous information from the bottom and top of the image. (ibid., line 109).

2. Final Model Architecture

The final model architecture consisted of a convolution neural network with the following layers and layer sizes:

1. Convolutional layer (with RELU) – 5x5 – Depth: 24 followed by Max pooling layer
2. Convolutional layer (with RELU) – 5x5 – Depth: 36 followed by Max pooling layer
3. Convolutional layer (with RELU) – 5x5 – Depth: 48 followed by Max pooling layer
4. Convolutional layer (with RELU) – 3x3 – Depth: 64 followed by Max pooling layer
5. Convolutional layer (with RELU) – 3x3 – Depth: 64 followed by Max pooling layer
6. Three (3) fully connected layers

Training Strategy

The network configuration, as mentioned earlier, is based on the seemingly simple architecture NVIDIA has used in [1]. This model was the final one chosen after many iterations and combinations as no combination of convolution layers, kernel sizes, depths, dropouts and pooling were producing satisfactory results, i.e., the car would eventually veer off the track.

Attempts to use Dropout layers resulted in lower accuracy in this network configuration, before and after flattening, and were thus dropped! The existing configuration was arrived at after many iterations.

The model was tested by running it through the simulator and ensuring that the vehicle stayed on the (simple) track.

As an “Adam Optimizer” was used (ibid., line 141), the learning rate was not tuned. Batch size and # of epochs, however, were.

Track and Simulation Images

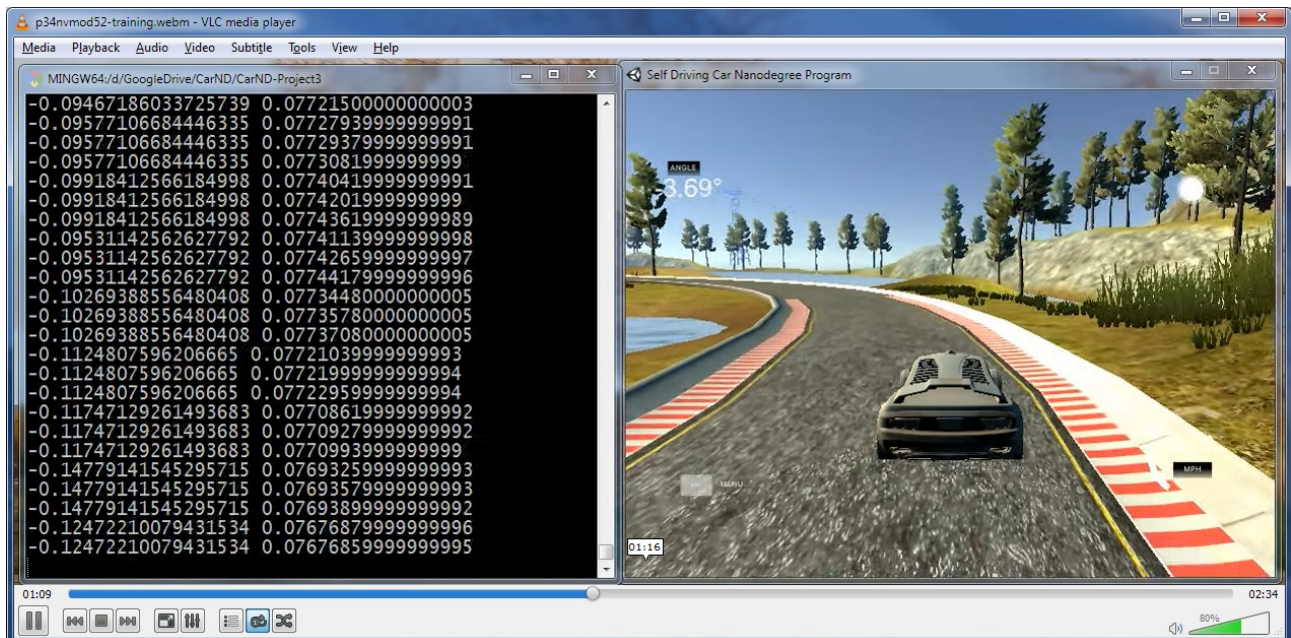
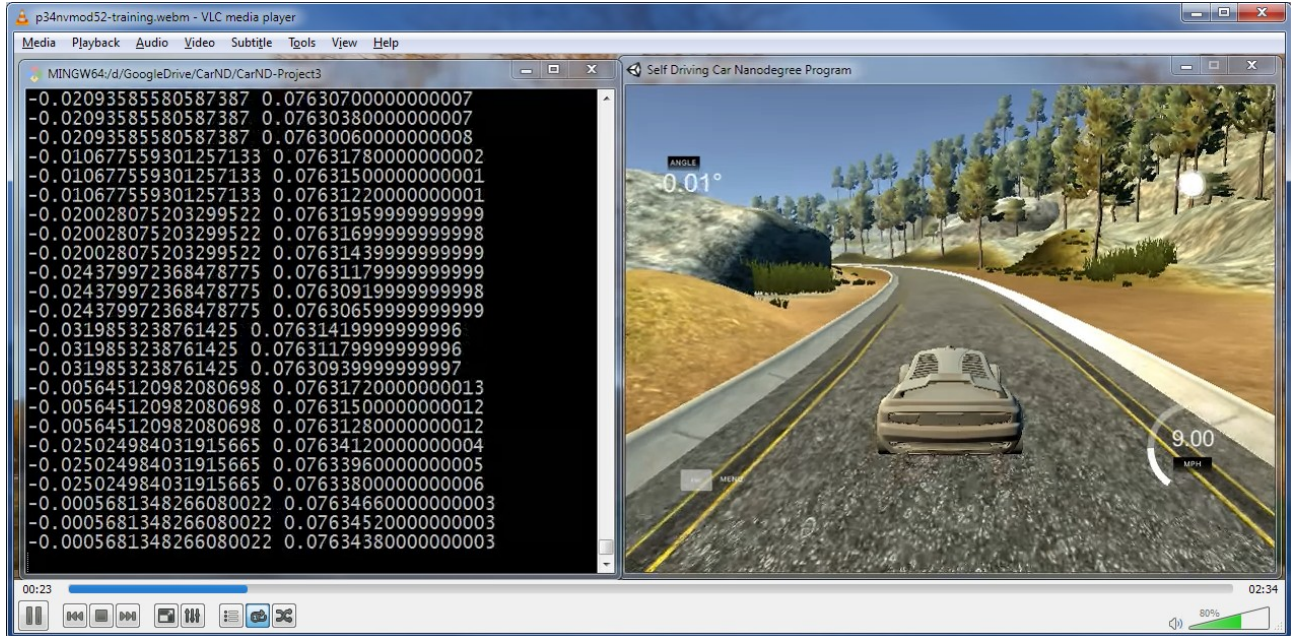
The provided training data (sample Udacity-provided Track 1 data) was chosen to train the network, and split into training/validation samples (80/20).

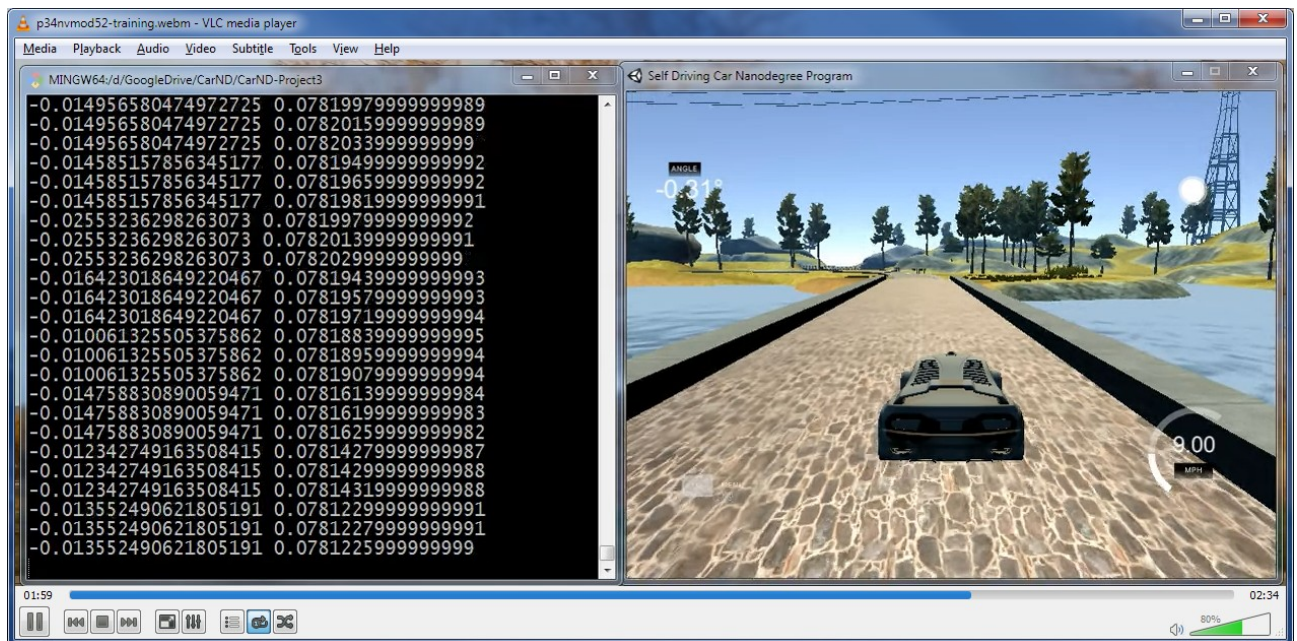
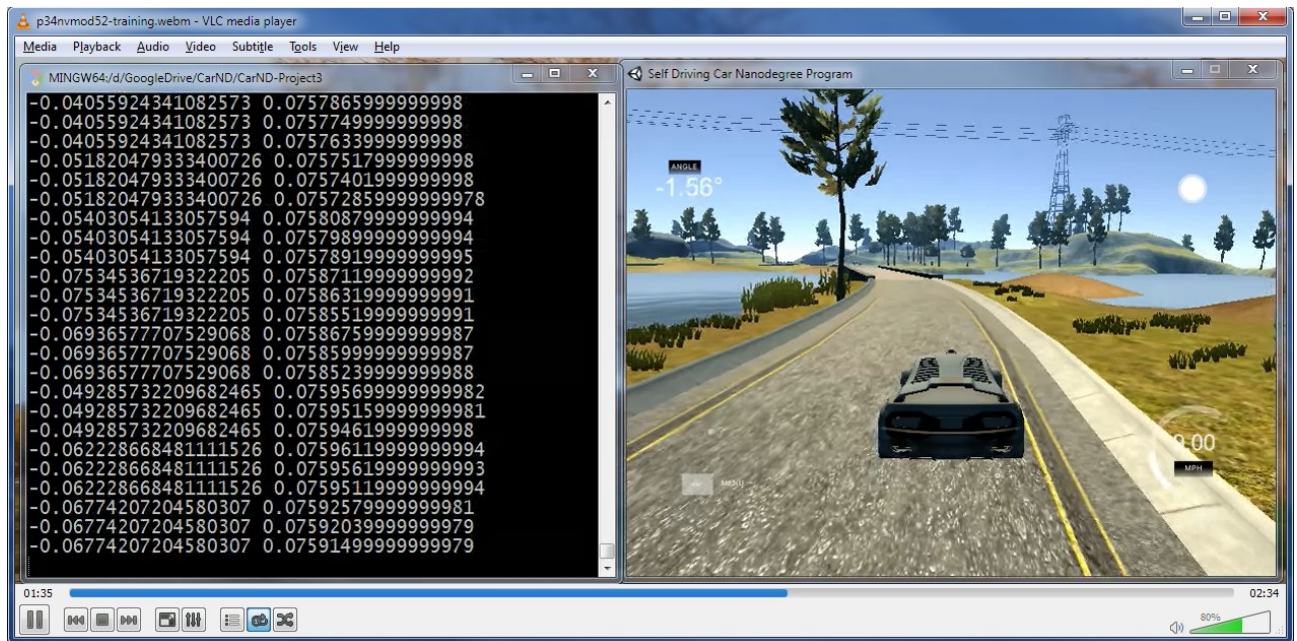
TRACK 1 IMAGES

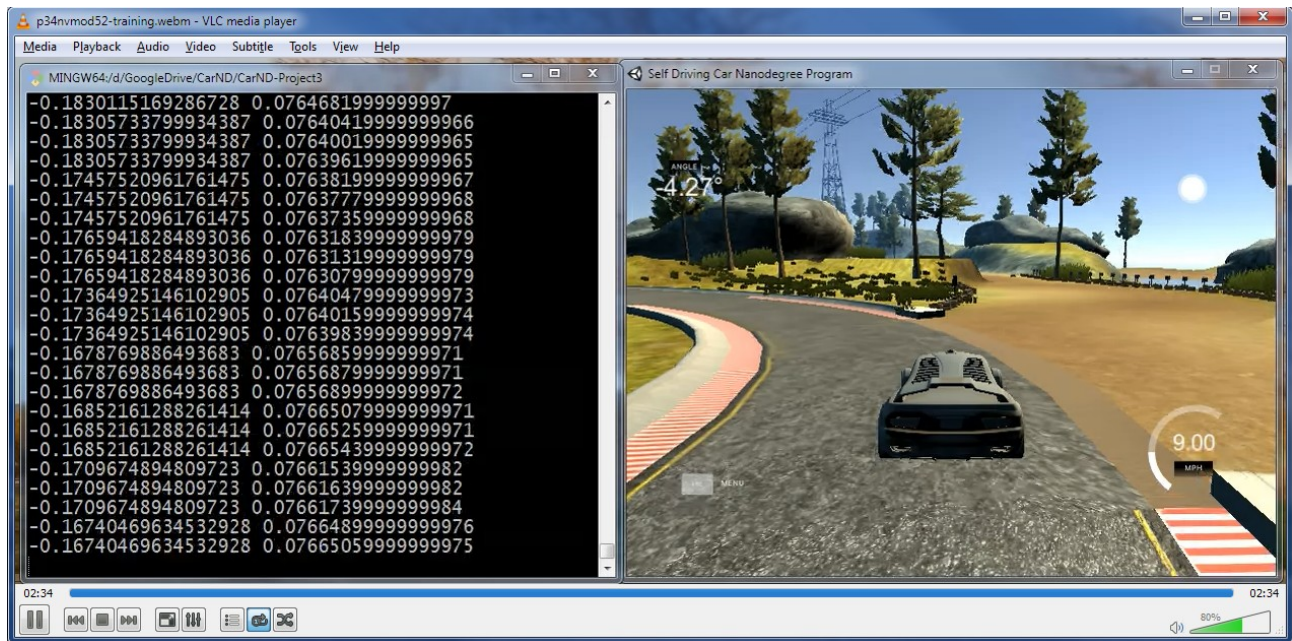


SCREENSHOTS OF THE SIMULATION

The final step was to run the simulator to see how well the car was driving around track one. The vehicle is able to drive autonomously around the track without leaving it.







Files Submitted

1. Files in the submission include

Link to the [project code](#)

- | | |
|---|----------------------------------|
| 1. <i>P34nvmod52.py</i> (File 1) | 2. <i>drive.py</i> (File 2) |
| 3. <i>P34nvmod52.h5</i> (File 3) | 4. <i>P3-Report.pdf</i> (File 4) |
| 5. <i>P3-SimulationVideo.webm</i> (plays in VLC Media Player) | |

2. To execute autonomous driving simulation

Prompt:> `python drive.py p34nvmod52.h5`

The Udacity Simulator should be started first. (Default Windows Desktop 64-bit.exe).

REFERENCE

1. arXiv:1604.07316v1 [cs.CV] 25 Apr 2016

["End to End Learning for Self-Driving Cars", Mariusz Bojarski, et al, NVIDIA Corp., Holmdel, NJ 07735]