Udacity Self-Driving Car Engineer Nanodegree

Behavioral Cloning

Introduction

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric points

Here I will consider the rubric points (https://review.udacity.com/#!/rubrics/432/view) individually and describe how I addressed each point in my implementation.

Writeup

This report is the project writeup.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- Behavioral_Cloning_report.pdf summarizing the results

- run1.mp4 containing a sample video of driving autonomously on test track 1
- run2.mp4 containing a sample video of driving autonomously on test track 2

2. Submission includes functional code

Using the Udacity provided simulator, drive.py file, and my saved model in model.h5, the car can be driven autonomously around the track by executing `python drive.py model.h5`. The file drive.py is unchanged from what was provided by Udacity.

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works. I utilized a Python generator in model.py with the goal of reducing memory usage.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model (model.py lines 57-74) consists of a five-layer convolutional neural network with 3x3 and 5x5 filter sizes and depths between 24 and 64 , followed by five fully connected layers.

The model includes RELU layers to introduce nonlinearity (code lines 66-72), and the data is normalized in the model using a Keras lambda layer (code line 58).

2. Attempts to reduce overfitting in the model

Each of the fully connected layers is followed by a dropout layer (code lines 66-74) in order to reduce overfitting. The data is split into training and validation sets (code lines 29, 52, 53) in order to avoid overfitting to the training set. The model contains dropout layers in order to reduce overfitting (model.py lines 21).

The model was finally tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an Adam optimizer, so the learning rate was not tuned manually (model.py line 77). Adam is a commonly used stochastic optimizer that adjusts the learning rate dynamically for each parameter. I implemented multiple dropout layers with a lesser dropout rate (0.3) instead of relying on a single dropout with a higher rate.

I also experimented with different batch sizes for the generators and 32 seemed to result in fastest training times.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road, and recording additional data for the most difficult parts of the track, e.g. turns, sand boundaries, bridges. For details about how I created the training data, see the next section.

Architecture and Training Documentation

1. Solution Design Approach

I started by developing a minimum viable behavioral cloning pipeline, and then improved it. This strategy seems necessary as it is not possible to test autonomous driving before all of the pipeline is working. The pipeline has following parts:

Data collection and preprocessing	Collecting data through driving the car in the simulator, recording different driving scenarios. Building a Python generator reading this data from the disk and feeding to model training with Keras. Cropping the top and the bottom parts of the images to reduce noise. Normalizing the RGB values

	to have a mean of 0 and variance of 1.
Defining the model	Using Keras to define a multi-layer CNN model.
Training the model	Training the model. I utilized the Udacity GPU workspace as the training was impossibly slow on CPU.
Testing in the simulator	Testing how well the simulator's autonomous driving mode works utilizing the model. Is the car able to stay on road and make sensible steering choices. I mainly used my own laptop for this test phase: running drive.py and the simulator locally on the laptop. The code in drive.py takes the created model.h5 as an argument.

I first implemented a very simple CNN model consisting of just two CNN layers and three fully connected layers like demonstrated on the course lecture. The model worked reasonably well and got the autonomous car through most of the test track 1. However, I moved on to build a more powerful model adapted from the autonomous vehicle team from Nvidia (*Keras Implementation of End to End Learning for Self-Driving Cars* by Baris Kayalibay, Grady Jensen, Patrick van der Smagt https://arxiv.org/abs/1604.07316). It was necessary to slightly modify the model architecture as I was utilizing different size image files. I also added dropout layers to reduce overfitting.

The data collection process is further discussed later in this report. I started with a base data set, and then moved on to collect more data on challenging situations.

2. Final Model Architecture

The final model architecture (model.py lines 57-74) consisted of a convolution neural network with the following layers and layer sizes.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 160, 320, 3)       0
_________________________________________________________________
```

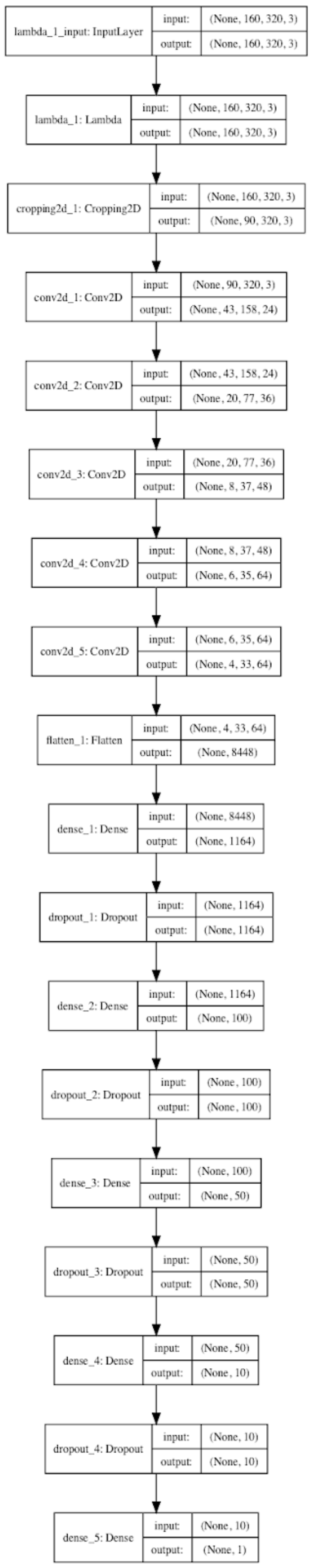
```
cropping2d_1 (Cropping2D)     (None, 90, 320, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)             (None, 43, 158, 24)       1824
_________________________________________________________________
conv2d_2 (Conv2D)             (None, 20, 77, 36)        21636
_________________________________________________________________
conv2d_3 (Conv2D)             (None, 8, 37, 48)         43248
_________________________________________________________________
conv2d_4 (Conv2D)             (None, 6, 35, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)             (None, 4, 33, 64)         36928
_________________________________________________________________
flatten_1 (Flatten)           (None, 8448)              0
_________________________________________________________________
dense_1 (Dense)               (None, 1164)              9834636
_________________________________________________________________
dropout_1 (Dropout)           (None, 1164)              0
_________________________________________________________________
dense_2 (Dense)               (None, 100)               116500
_________________________________________________________________
dropout_2 (Dropout)           (None, 100)               0
_________________________________________________________________
dense_3 (Dense)               (None, 50)                5050
_________________________________________________________________
dropout_3 (Dropout)           (None, 50)                0
_________________________________________________________________
dense_4 (Dense)               (None, 10)                510
_________________________________________________________________
dropout_4 (Dropout)           (None, 10)                0
_________________________________________________________________
dense_5 (Dense)               (None, 1)                 11
=================================================================
Total params: 10,088,055
Trainable params: 10,088,055
Non-trainable params: 0
```

First, I used a Lambda function to normalize the RGB image. Then, I cropped pixels from the top and the bottom of the image to reduce unnecessary noise.

Then, I used a series of convolutional layers. The layers conv2d_1, conv2d_2 and conv2d_2 are results of applying a 5x5 filter with valid padding. The rest of the convolutional layers are results of applying a 3x3 filter, also with valid padding.

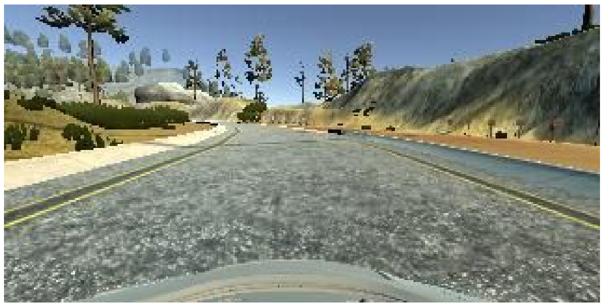
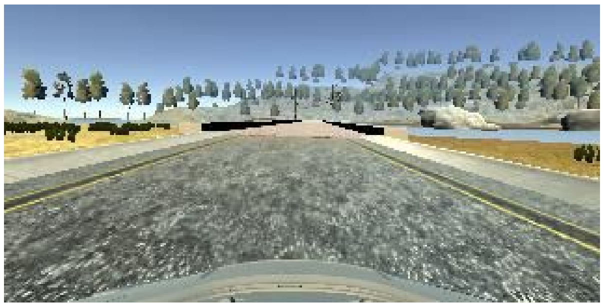
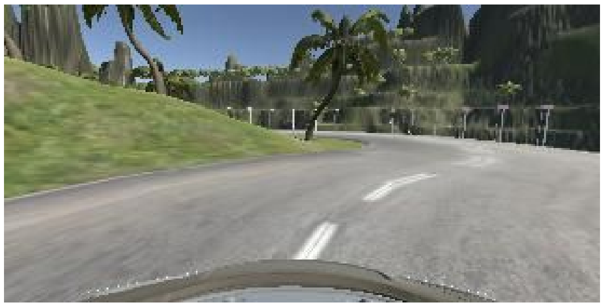
Finally, I flattened the data into a one-dimensional vector (code line 65) and used a series of fully connected layers and dropout layers (code lines 65-74). All fully connected layers use ReLU activation for non-linearity.

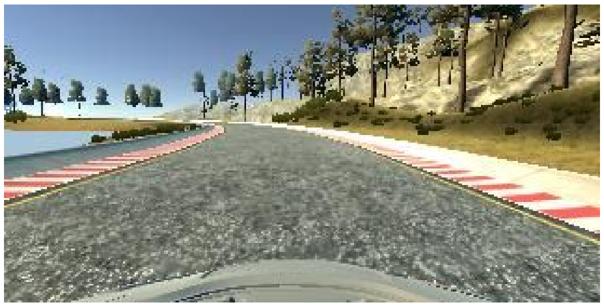
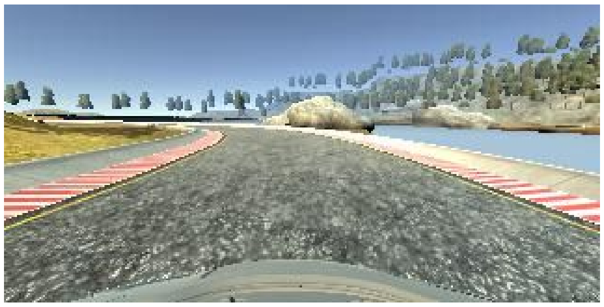
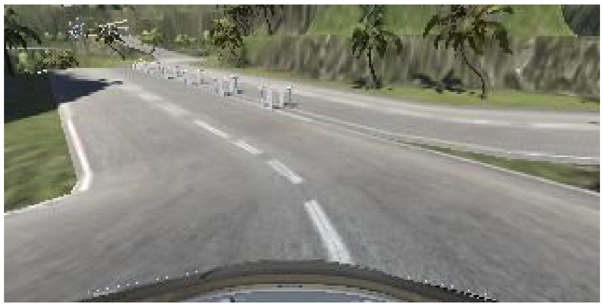
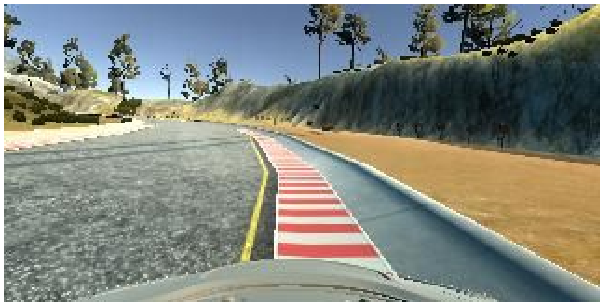
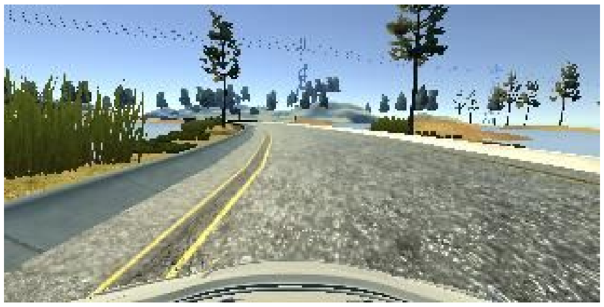
The end result is the model predicting the steering angle as a single number (code line 74).


lambda_1_input: InputLayer
input: (None, 160, 320, 3)
output: (None, 160, 320, 3)
lambda_1: Lambda
input: (None, 160, 320, 3)
output: (None, 160, 320, 3)
cropping2d_1: Cropping2D
input: (None, 160, 320, 3)
output: (None, 90, 320, 3)
conv2d_1: Conv2D
input: (None, 90, 320, 3)
output: (None, 43, 158, 24)
conv2d_2: Conv2D
input: (None, 43, 158, 24)
output: (None, 20, 77, 36)
conv2d_3: Conv2D
input: (None, 20, 77, 36)
output: (None, 8, 37, 48)
conv2d_4: Conv2D
input: (None, 8, 37, 48)
output: (None, 6, 35, 64)
conv2d_5: Conv2D
input: (None, 6, 35, 64)
output: (None, 4, 33, 64)
flatten_1: Flatten
input: (None, 4, 33, 64)
output: (None, 8448)
dense_1: Dense
input: (None, 8448)
output: (None, 1164)
dropout_1: Dropout
input: (None, 1164)
output: (None, 1164)
dense_2: Dense
input: (None, 1164)
output: (None, 100)
dropout_2: Dropout
input: (None, 100)
output: (None, 100)
dense_3: Dense
input: (None, 100)
output: (None, 50)
dropout_3: Dropout
input: (None, 50)
output: (None, 50)
dense_4: Dense
input: (None, 50)
output: (None, 10)
dropout_4: Dropout
input: (None, 10)
output: (None, 10)
dense_5: Dense
input: (None, 10)
output: (None, 1)

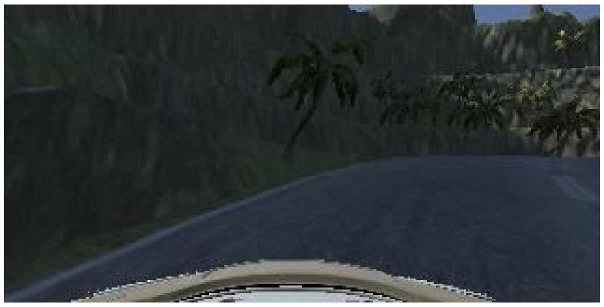
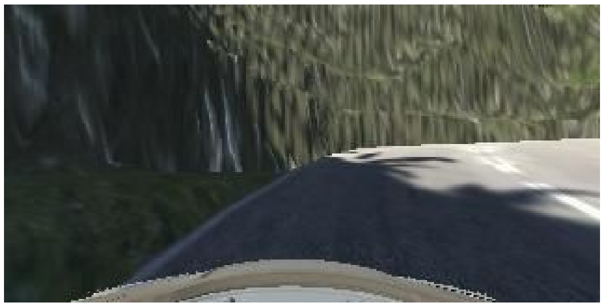
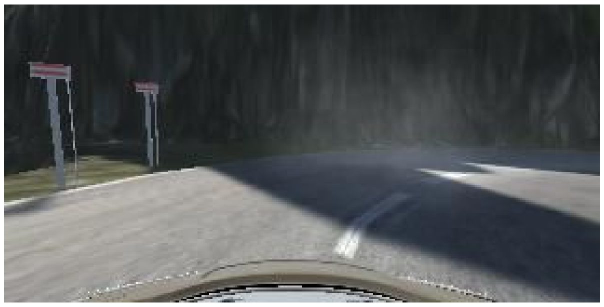
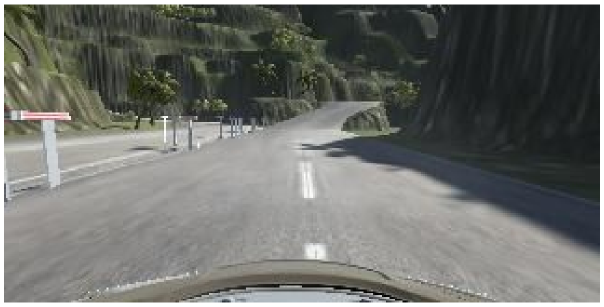
3. Creation of the Training Set & Training Process

I started by recording the base data set of driving on both tracks, added with reverse driving for generalization and repeat driving in challenging road sections (e.g. turns, bridges) to have more data on these special conditions. After initial tests, I also recorded recovery driving about the car recovering back to the center of the road when it had drifted to left or right. Find the list of the collected data below.

Track 1 normal driving	
Track 1 reverse driving	
Track 2 normal driving	

Track 1 careful turn and bridge driving	
Track 1 reverse careful turn and bridge driving	
Track 2 reverse driving	
Track 1 recoveries from right to center It is important to record only the time when the recovery is happening, not the act of driving car to the side, which would teach the model to randomly drift to the side	
Track 1 recoveries from left to center It is important to record only the time when the recovery is happening, not the act of driving car to the side, which would teach the model to randomly drift to the side	

After testing a model trained with the base data set, I observed problems especially with the shady sections of the track 2, and decided to record more data on the challenging sections.

Track 2 careful driving in shady road sections	
Track 2 driving through challenging shade patterns	
Track 2 driving through challenging shade patterns	
Track 2 driving through challenging shade patterns	

At this point the data set was already close to 10,000 steering measurements and because training the model slowed down significantly, I decided against using data augmentation. But, data augmentation could be used to accelerate the data collection process, as a similar-size data set could be built from fewer samples. For the same reason, I also only used the center camera data.

Simulation

1. Car is able to navigate correct on test data

With the final model I built, the car is able to navigate the test track 1 with a very good driving behavior, and without ever drifting over the lane lines. The model works to an extent on the test track 2, but eventually the various different road borders and shadow patterns are too much for the it.

Improvement ideas

The model worked very well on the easy test track 1, but even there I would not yet trust it with human passengers. I would work on following areas to improve the model:

- **Higher quality training data**: Driving the car manually in the simulator was somewhat challenging. Controlling it on keyboard resulted on choppy steering patterns (and controlling it on a laptop trackpad was nearly impossible). I would experiment collecting the test data using an actual steering wheel input device for more realistic and higher quality steering measurements.
- **More data on shady roads:** The patterns drawn by shadows were very hard for the model to differentiate from actual road borders. More data on these tricky sections would be beneficial.
- **Thresholding techniques:** The challenge with shady road conditions could also be helped with thresholding techniques highlighting the lane lines and road signage.

Due to the fact that increasing the model complexity from the initial simple CNN model to the Nvidia model resulting only moderate improvements, I would primarily focus on the areas above before pursuing an even more complex model. Complex models are slower to train, require more computing to run, and require more storage capacity.