Writeup Report – Behavioral Cloning

Name: Yongxu Yao

Behavioral Cloning Project

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 individually and describe how I addressed each point in my implementation.

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
- writeup_report_behavioral_cloning.pdf summarizing the results

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.h5
```

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.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model is inspired by NVIDIA's CNN architecture (reference: https://devblogs.nvidia.com/deep-learning-self-driving-cars/). The model consists of 5 convolutional layers, followed by four fully connected layers. (line 68 to 87 in model.py) The first convolutional layers used 2×2 stride and a 5×5 kernel, and the following two convolutional layers are non-strided, with 3x3 kernal size. The four fully-connected layers leading to a signal final output with the prediction of steering angle.

To make the training more efficient, at the very beginning of the network, the x data are normalized with 0 mean, and then the image is cropped to delete vehicle hood and environmental data, keeping only the road information. (line 63 to 65)

2. Attempts to reduce overfitting in the model

The model uses dropout layers after every layers, to reduce overfitting. The epoch is set to 25 so that the model get adequate training not overfitting. Also, the model was 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 90).

4. Appropriate training data

Training data was carefully chosen to keep vehicle on the track. I collected following data:

1. Normal driving counterclockwise. (2 laps)
2. Normal driving clockwise. (1 lap)
3. Recover vehicle from road side. (about 10 set)
4. Normal driving in the second map. (half lap)

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy here is to make components works first, in a simple way. Then I changed certain components to more advanced ones for better performance.

My first step was to use a very simple network which only have one fully connected layer to test my environment working or not. The model output the correct model.h5 file which can drive the car a little bit in the simulator.

Then I change the model to LeNet, and collected some more data. The car was able to pass the first corner with this architecture and then drove into water. Since the LeNet seems working, I turned to the data side. I added the left and right camera image to X_train, with corresponding y_train adding +0.2 or -0.2. (line 21) Also, all images were flipped to create 2 times more data. By here, I utilized 6x more image data than the original.

The car passed the second corner and bridge with the augmented data. Then I applied a more powerful model, which is NVIDIA's CNN model. To prevent overfitting, I used droupout after each convolutional layer. The car drove better with this model, but it still drove off road in some spot.

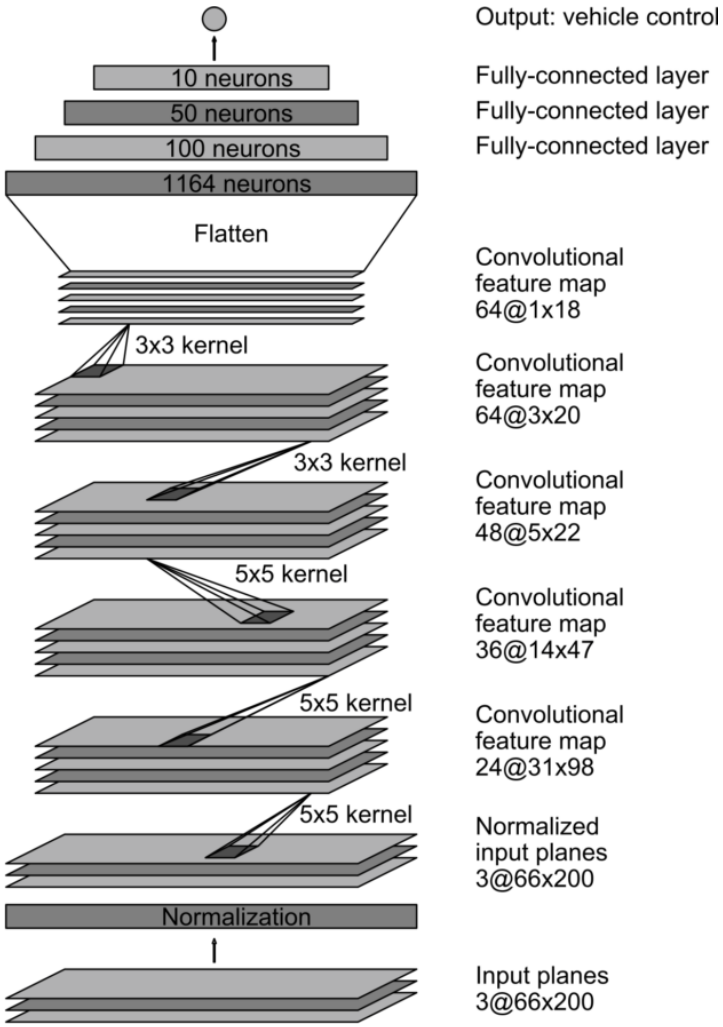
Then I collected more data, where I pulled back the car from the road side. I also recorded half lap in the second lap to bring some diversity to the data. With more data, the model finally did a good job. The vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py line 59-87) consisted a convolution neural network with the following layers:

1. Lambda layer to normalize input value
2. Cropping2D layer to crop image, only keeping useful information
3. Three convolutional layers with 2×2 stride and a 5×5 kernel, each followed by Dropout with dropping probability of 0.1
4. Two convolutional layers with 3×3 kernel, each followed by Dropout with dropping probability of 0.1
5. Four fully connected layers, each followed by Dropout with dropping probability of 0.1

Here is a visualization of the architecture (from NVIDIA):



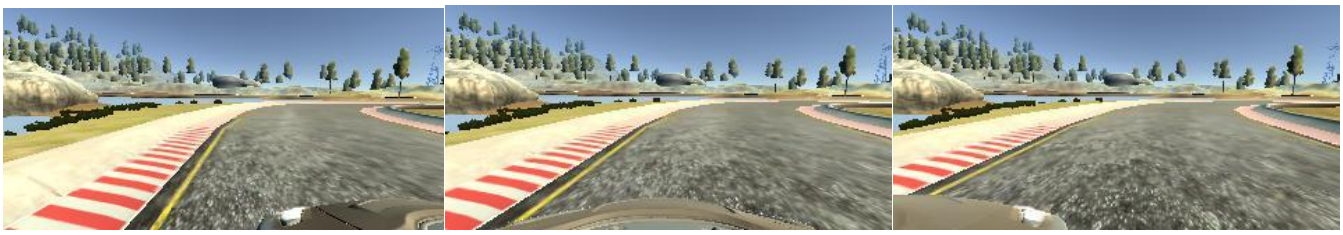
3. Creation of the Training Set & Training Process

Some of the training process are discussed in the design approach section. The final data I used is a 6x more data then using only the middle camera. The middle, left and right image I used is like:

Image(left) Image(middle) Image(right)

Each image above will be flipped and write back to data set, which creates 2x more data.

In the data set I also have around 10 rounds of recovering the vehicle from road side:

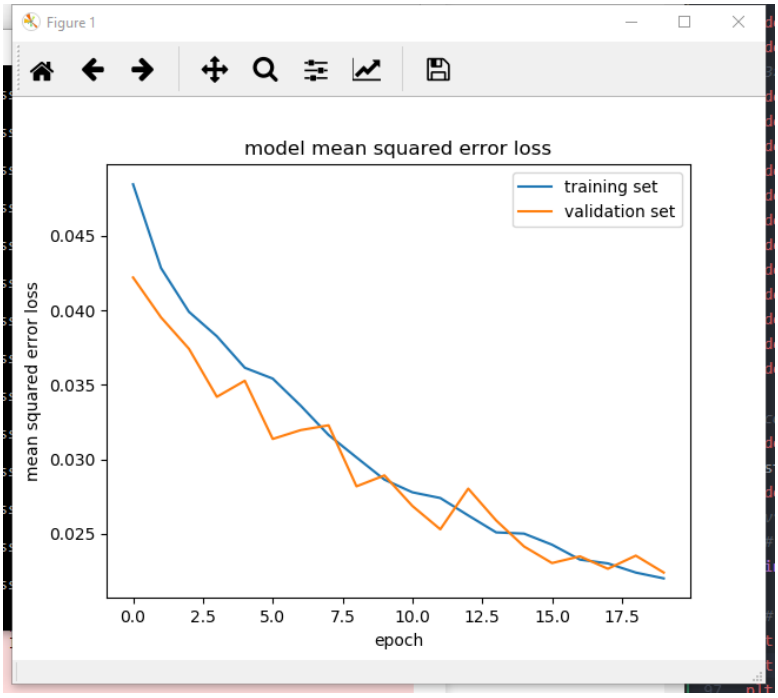


And to make the dataset more diverse, I also recorded on the second track:

The final original testset contains 19722 images. After augmentation, the total X_all is 39444. Here is the number of sample for different sets:

dataset	number
traininig	28399
validation	7099
test	3944

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 25 as evidenced by the following chart. When epoch greater than 25, the training accuracy is always higher than validation, and both not improving, which is a indication that the model is overfitting over 25 epochs.



I used an adam optimizer so that manually training the learning rate wasn't necessary.

Improvement

1. The car swing back and forth on the track, which I plan to add some smoothing filters to it.
2. Try collect data based on track 2 and try the model on track 2.