Behavioral Cloning

Behavioral Cloning Project

Attention: I wrote what I changed from my first submission in red.

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 Kera's 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.md or writeup_report.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

Differences from the previous submission are

1. To collect more smooth driving data at the corners of the track one.
2. To make the brightness augmented images from the original images.
3. To add dropout layer in the CNN.

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 5x5 and 3x3 filter sizes and 24, 36, 48, and 64 depths. (model.py lines 173, 178, 181, 184, and 187)

The model includes RELU layers to introduce nonlinearity (model.py lines 173, 178, 181, 184, and 187), and the data is normalized in the model using a Keras lambda layer (model.py lines 168).

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py lines 36). 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 lines 200).

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 the smooth drive at the curves.

I'll show you how I created the training data at the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to collect enough and appropriate driving data and have the model learned by their data.

My first step was to use a convolution neural network model similar to the NVIDIA's architecture featured in the lesson. I chose this model because LeNet architecture, which I used at first, was not enough and I thought I needed more powerful neural network.

The next step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track. To improve the driving behavior in these cases, I used the left and right camera images, the mirror images of my driving data, a bit shifted images from the original, and the brightness augmented images. In addition, I collected and added the images of the smooth run at the corners in track one.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

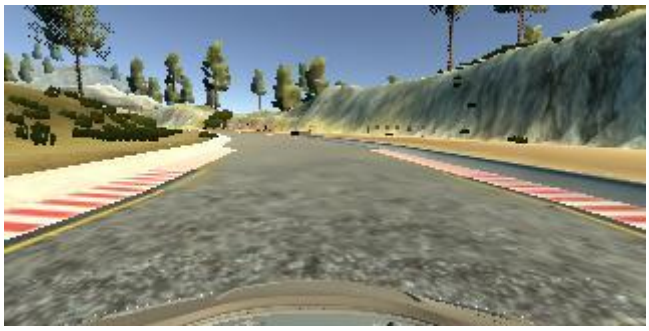
2. Final Model Architecture

The final model architecture (model.py lines between 164 and 195) consisted of a convolution neural network with the following layers and layer sizes .

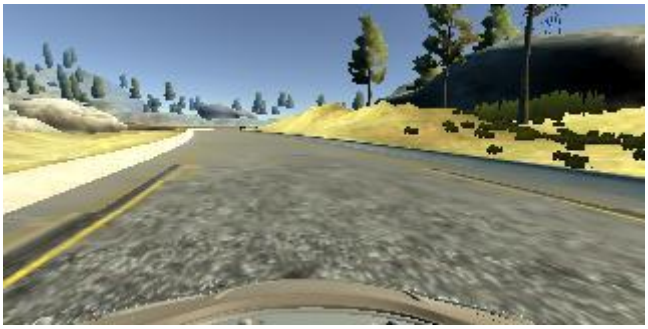
Layer	Description
Input	65x320x3 image
Convolution 5x5	2x2 stride, valid padding, outputs 31x158x24
RELU	
Dropout	Keep probability = 0.5
Convolution 5x5	2x2 stride, valid padding, outputs 14x77x36
RELU	
Convolution 5x5	2x2 stride, valid padding, outputs 5x37x48
RELU	
Convolution 3x3	1x1 stride, valid padding, outputs 3x35x64
RELU	
Convolution 3x3	1x1 stride, valid padding, outputs 1x33x64
RELU	
Flatten	Output 2112
Fully connected	Output 100
Fully connected	Output 50
Fully connected	Output 1

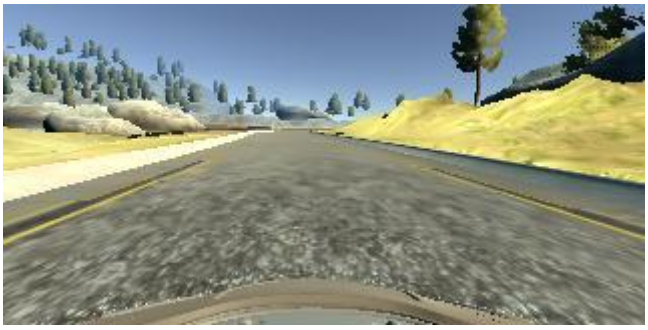
3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:



I then recorded the vehicle recovering from the left side and right side of the road back to center so that the vehicle would learn to go back from the both sides to center. These images show what a recovery looks like starting from the left side of the road. :





To augment the data set, I also flipped images and angles thinking that this would make model more robust for the reverse move of the recorded data. For example, here is an image that has then been flipped:

In order to enhance the behavior at the corners, I collected the slow and smooth curve data one lap.

After the collection process, I had 37,105 number of data points. (They include the flipped images.) I then preprocessed this data by cropping and normalization.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

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 3 because the value loss didn't get low any more when I increased the number of epochs. I used an adam optimizer so that manually training the learning rate wasn't necessary.