Write-up Report – Behavioral Cloning Project

Self-Driving Car Nano Degree Program – Term1, Project3

###Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

---

1. Files Submitted & Code Quality

1.1. 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.pdf or writeup_report.pdf summarizing the results

1.2. 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
```

1.3. 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.

2. Model Architecture and Training Strategy

2.1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 5x5 & 3x3 filter sizes and depths between 24 and 64 (model.py lines 82-94)

The model includes RELU layers to introduce nonlinearity (code lines 82,85,88,91 & 94), and the data is normalized in the model using a Keras lambda layer (code line 79).

The model includes a cropping layer (code line 77) to trim image in order to see only the road section

2.2. Attempts to reduce overfitting in the model

The model was trained and validated on different data (forward driving, reverse driving, recovery lap driving etc.,) sets to ensure that the model was not overfitting. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

2.3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 103).

2.4. 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 reverse lap driving

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

3. Model Architecture and Training Strategy

3.1 Solution Design Approach

I used the following steps in order to arrive at a final solution

Step1: with Basic model to check the data processing and integration with simulator

First I started with a basic linear model model with only one layer. The objective is to test check the following.

- To check the correct behavior of data processing logic this includes reading lines from excel file, extracting of image names, reading images correctly
- To check whether the model is saved correctly
- To check the integration of model with simulator

After training the model and tried to drive in Autonomous mode, the car drove terribly wrong. Of course, it’s not a surprise as I already knew the model will under fit.

Step2: with Nvidia model and training only with center images from forward and reverse lap

- I wanted to try the architecture used by Nvidia for steering wheel prediction.
- I also added a cropping layer to trim the images in order to remove the unwanted information
- I used only the center camera images from forward lap driving and reverse lap driving for training as I wanted to check the driving behavior only with this data set.
- The Car drove most of the time properly. But could not able to recover when it went off-track

At this point I am convinced that the model architecture is good enough for the problem may be I need more dataset for training, especially when the car drives off-track

Step3: with Nvidia model and training with center, Left & Right images from forward, reverse lap and recovery lap

- I performed a recover lap driving to augment my dataset, especially for recovering during tight curves. I also used the left and right camera images for training.
- I used an offset of +/- 0.25 to the steering wheel angle for left & right camera images respectively (code lines 53 & 59)
- After training with this data set, the car was able to drive around the track without any deviations

3.2 Final Model Architecture

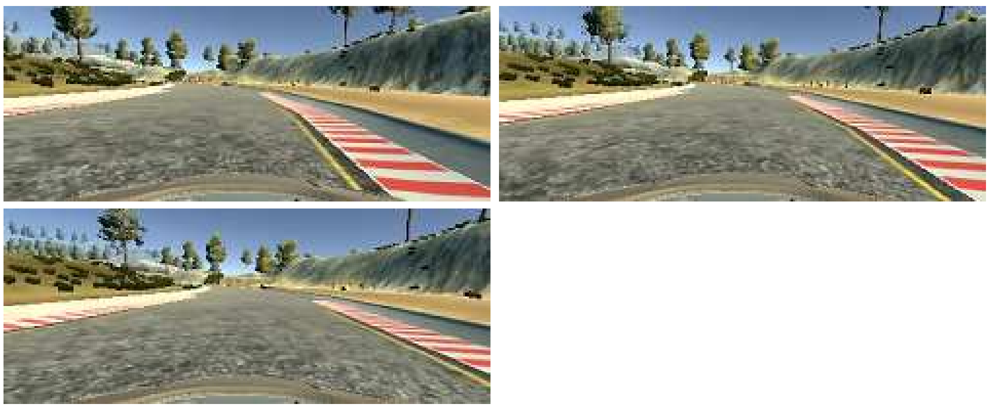
- As I already mentioned in section 3.1, I used Nvidia model architecture for this problem. Following is the architectecture.
- The model architecture is explained very well with comments in the model.py (lines 74 to 101)

4. Creation of the Training Set & Training Process

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

Then I recorded two laps of driving in the reverse direction as I will have more data for the right turns also

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover when it goes off-track. These images show what a recovery looks like starting from right edge of the lane to the middle of the lane



After the collection process, I had around 40,000 number of data points.

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. I trained the model for 10 Epochs, The training loss and validation loss were constantly decreasing. At the end of 10 Epochs the training loss was 0.0152 & validation 0.0238. As these loses almost reached 0, I decided to stop the training after 10 Epochs. I used an adam optimizer so that manually training the learning rate wasn't necessary.