Behavioral Cloning

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 (https://review.udacity.com/#!/rubrics/432/view) 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

1. An appropriate model architecture has been employed

My model is the same NVIDIA architecture mentioned in the lecture, which consists of 5 convolution layers and 4 dense layers. I found this works better than modification of the Lenet architecture.

My final model consisted of the following layers:

Layer	Description
Input	160x320x3 normalized image
Normalizing	lambda x: (x / 127.5) - 1.0
Cropping	size ((70,25), (0,0))
Convolution 5x5	2x2 stride, valid padding, outputs 24, with RELU activation
Convolution 5x5	2x2 stride, valid padding, outputs 36, with RELU activation
Convolution 5x5	2x2 stride, valid padding, outputs 48, with RELU activation
Convolution 3x3	1x1 stride, valid padding, outputs 64, with RELU activation
Convolution 3x3	1x1 stride, valid padding, outputs 64, with RELU activation
Flatten	
Fully connected	outputs 100
Fully connected	outputs 50
Fully connected	outputs 10
Fully connected	output 1 logits
Softmax	output 1 probability

2. Model parameter tuning and attempts to reduce overfitting

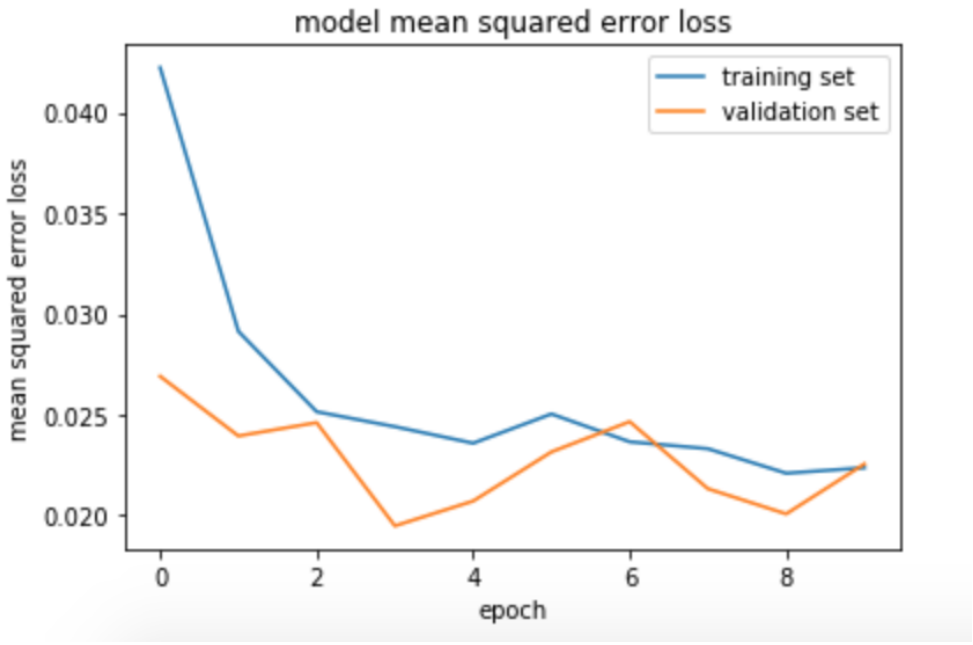
The model didn't worked well at the beginning so I added dropout layer but there wasn't much improvement. I found the key is my training data. It worked well with good training data and without dropout, so I ended up without dropout layers.

The model used an adam optimizer, so the learning rate was not tuned manually.

The model was trained and validated on different data sets to ensure that the model was not overfitting. For each data set, I always try 10 epochs first. Then I plot the loss on both training and validation set. Then I pick the point where the validation loss stop decreasing. Finally epoch=4 in my final model.

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

```
dict_keys(['loss', 'val_loss'])
```



4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. For all datasets used, I first flipped the images and times -1.0 to angle to make the training set contain equally many left and right turns so as to make the model less biased.

At first I only used the sample data provided by Udacity. But this worked poorly.

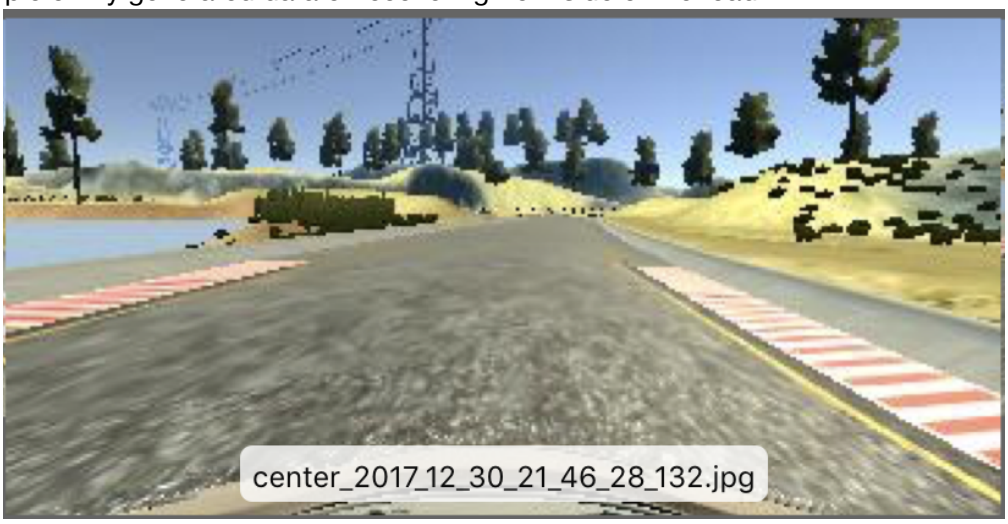
Then I have to generate training data myself. I first focused on center-lane driving, driving clockwise and counter-clockwise both for two laps. Then I combined this data set `myData` with the Udacity dataset `data`. This worked amazingly well. The car was able to keep driving on the road, except that large right turn after the bridge, where half of the car got outside the road, but was also able to get back on the road though.

Then I modified the model by adding dropout and adding side-cameras. But there are always some minor mistakes when I test it in the simulator. So I decided to generate more and better data. I drove more carefully at turns (more slowly and with more frequent minor turns). Adding side-cameras data and tested all datasets, I finally got the car to always stay on the road. Genrating data of recovering from the left and right sides of the road is very hard, and using side-cameras data definitely improved the model.

The dataset `moreTurns` includes a lot of turning data only. The dataset `moreData` includes 1 lap of clockwise center-lane driviing, 1 lap of counter-clockwise center-lane driving and a lot of recovering from side of the road driving.

In my final model that worked, I removed the data `data` from Udacity and the data set containing only turns (maybe too much turns) `moreTurns`, kept only the center-lane driving data `myData` and the data combining center-lane driving and a lot of recovering from side of the road `moreData`. When I tested on the simulator, the car isn't as stable as the original model when I only used center of the road driving especially on the bridge. This may because when I generate data of recovering from sides of the road, I also included a lot of "driving towards side of the road" data. This can be avoided by driving more carefully. But overall it's a good model, the car always stays on the road and can keep driving for hours.(I took a nap and it is still driving.)

Here is an example of my generated data of recovering from side of the road:



Here is an example of the car recovering from the side of the road in autonomous mode. The car is slightly turning right before it hits the bridge on the left.