Behavioral Cloning

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

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.pdf summarizing the results
- test.mp4 for the generated video

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 based on Nvidia network introduced in the lectures. First layer of the model is a Cropping2D layer, which crops the trees and car and only keeps the relevant lane portion. My model consists of a convolution neural network with 5x5, 3x3 filter sizes and depths between 24 and 64 (model.py lines 121-125)

The model includes RELU layers, defined in Conv2D, to introduce nonlinearity (121-125), and the data is normalized in the model using a Keras lambda layer (code line 120).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 128). I tried 0.2 and 0.5 dropout rates and then found that 0.2 was working better.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 136). 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. Setting the initial LR for optimizer is important. I tried different LRs(0.1, 0.01, 0.001, 0.0001) and found that the default(0.001) was working fine.(model.py line 135).

4. Appropriate training data

I only used the sample training data that was provided in the project resources. I tried a lot to generate and collect my own training data. Turns out I am terrible driver. I relied on the sample training data for training my model.

Architecture and Training Documentation

1. Solution Design Approach

The overall strategy for deriving a model architecture was to try out different models and pick the model with lowest loss.

First, I started with Resnet34 pretrained model. I removed the last layer and added some fully connected layers. Training was taking a lot of time because of high number of parameters. Then, I thought of using the Nvidia model for training. I had noticed that it had very few parameters, so I thought of giving it a try.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that the Nvidia model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I modified the model by adding dropout layers after the fully connected layers (except the last one). I tried 0.5 and 0.2 dropout rates. 0.2 dropout rate was working really well.

Then I trained the model with different learning rates. Although Adam adjusts the learning rate, setting initial learning rate is important. I tried 0.1, 0.01, 0.001, 0.0001 learning rates. 0.001 was performing the best, since it's the default learning rate for Adam, I didn't define any learning rate.

The final step was to run the simulator to see how well the car was driving around 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 118-133) consisted of a convolution neural network with the following layers and layer sizes:

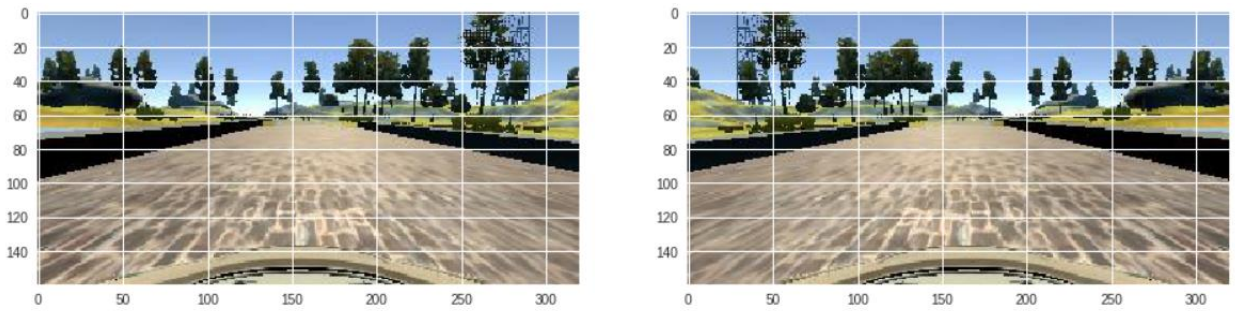
Layer (type)	Output Shape	Param #
cropping2d_1 (Cropping2D)	(None, 80, 160, 3)	0
lambda_1 (Lambda)	(None, 80, 160, 3)	0
conv2d_1 (Conv2D)	(None, 38, 78, 24)	1824
conv2d_2 (Conv2D)	(None, 17, 37, 36)	21636
conv2d_3 (Conv2D)	(None, 7, 17, 48)	43248
conv2d_4 (Conv2D)	(None, 5, 15, 64)	27712
conv2d_5 (Conv2D)	(None, 3, 13, 64)	36928
flatten_1 (Flatten)	(None, 2496)	0
dense_1 (Dense)	(None, 100)	249700
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 50)	5050
dropout_2 (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 10)	510
dropout_3 (Dropout)	(None, 10)	0
dense_4 (Dense)	(None, 1)	11

Total params: 386,619
Trainable params: 386,619
Non-trainable params: 0

3. Creation of the Training Set & Training Process

I only used the sample training data that was provided in the project resources. I tried a lot to generate and collect my own training data. Turns out I am terrible driver. I relied on the sample training data for training my model.

To augment the data sat, I also flipped images and angles thinking that this would compensate for the smaller number of right turning curves. For example, here is an image that has then been flipped:



I finally randomly shuffled the data set and put 20% of the data into a validation set. Within the training generator, I used left and right images as well by applying a correction factor of 0.2 on angles.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. I ran the model for 10 epochs and noticed that the model was underfitting. So, I trained the model for 30 epochs, where I noticed the validation loss was oscillating in a range, so I stopped training further. I used Adam optimizer with default learning rate.