Behavioral Cloning

How to train your Car

We will utilize Deep Learning techniques to predict the angle of car turn and compare it with ground truth and train over iterations to have model learn right weights. Once the weights are set with minimal error, we will let the car loose on a track to see how it performs in autonomous mode.

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

Implementation

Here I will consider 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 consists of a convolution neural network with 3x3 filter sizes and depths between 32 and 128 (model.py lines 18-24)

The model includes RELU layers to introduce nonlinearity (code line 20), and the data is normalized in the model using a Keras lambda layer (code line 18).

2. Attempts to reduce overfitting in the model

- The model contains dropout layers to reduce overfitting (model.py lines 57 & 63).
- I also collected the data over 2 laps to generalize the model better
- I flipped each image in the data set to generalize over left and right turn and not overfit only for one type of turn

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 70).

4. Appropriate training data

Collecting training data was the hardest part. I got the feeling that the way you drive in training mode affects how your car reacts in autonomous mode. Keeping car in the center of the track and not making many turns is not a good way of training. I felt that constantly taking right and left turns helped greatly in making decisions in autonomous mode; especially training on lot of data where car is coming inside towards the center from the boundary. Also, training for different types of boundaries helped a lot. Otherwise the car won't certain type of boundaries. Training data was eventually chosen to keep the vehicle driving on the road.

16	Center Image	Left Image	Right Image	Steering Angle	Throttle	Break	Speed
17	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0	0	21.76894
18	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0.219383	0	21.68102
19	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0.400436	0	21.76183
20	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0.618471	0	22.08006
21	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0.857783	0	22.62736
22	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	1	0	23.19052
23	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	1	0	23.96658

Model Architecture and Training Strategy

1. Solution Design Approach

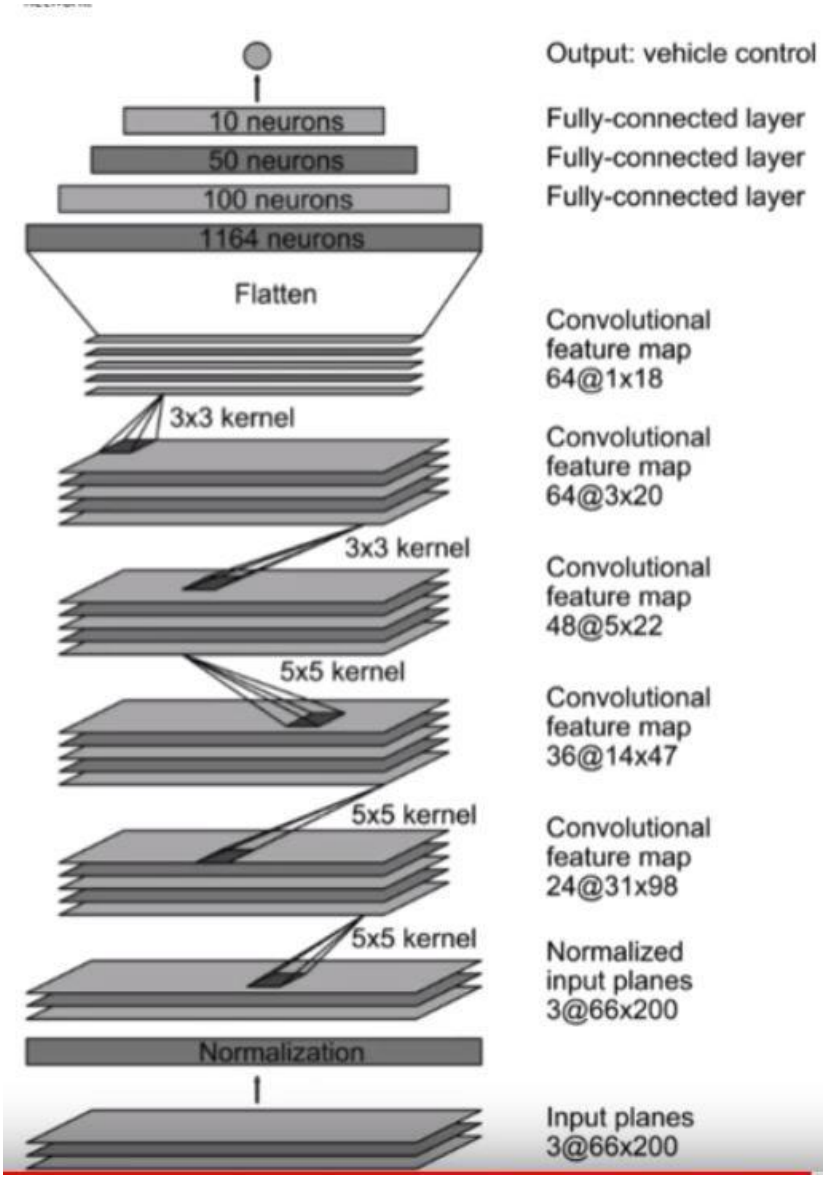
The overall strategy for deriving a model architecture was to have the lowest delta between training and validation runs. To gauge how well the model was working, I split my image and steering angle data into a training and validation set.

- My first step was to use Flatten and Dense layer to see how a very basic model performs. The result was very poor as expected and the car didn't even run and mse was very high for both testing and validation set.
- My second approach was to use Convolution layer and use 6 filters and 5x5 kernels with 'relu' as the activation layer. I also used a lambda function to perform image normalization on my training data set. The mse training and validation became lower but looked like i will still have very far to go. Difference between training and validation mse was also high indicating overfitting.
- I used the Nvidia's neural network architecture which has 5 Convolution2D layers and 3 FC layers. When I started training with this, the mse dropped a lot indicating this is a good network choice here. The delta between Training and Validation was also very less indicating that my model is now generalizing.

I found that my first 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. At the end of the process, the vehicle can drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture is provided in model.py file which is very close to Nvidia's architecture:



3. Creation of the Training Set & Training Process

I believe creating the right training data set for this project was the biggest task.

To capture good driving behavior, I first recorded two laps on track one using center lane driving. 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 recognizes the edges and turn towards center. I also collected data for different types of boundaries as my initial data didn't capture lots of images for each type of boundaries in track. Then I repeated this process on track two to get more data points.

To augment the data sat, I also flipped images and angles using OpenCV flip API. This helped a lot as the circuit was a loop and this rightly generated a healthy data set for both left and right turn.

I also cropped the images; 70 pixels from top and 25 from bottom so that my model learns only on the features it needs to and dropping other unnecessary features in the image.

I finally randomly shuffled the data 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 were 10.