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

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

```
sh
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 24 and 64 (model.py lines 216-229)

The model includes RELU layers to introduce nonlinearity (code line 219), and the data is normalized in the model using a Keras lambda layer (code line 217). A cropping layer is added to minimize additional pixels (code line 218).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 225, 227).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 35-148). 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 line 231).

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.

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

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was based on the original Nvidia model for driving behavior cloning.

My first step was to use a set of 4 convolution neural layers followed by 3 fully connected layers similar to the Nvidia approach. I thought this model might be appropriate because Nvidia has succeed on this model so it should be complicated just enough.

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 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.

To combat the overfitting, I added to dropout layers (50% dropout) after each dense layers so that they can eliminate any possible over fittings.

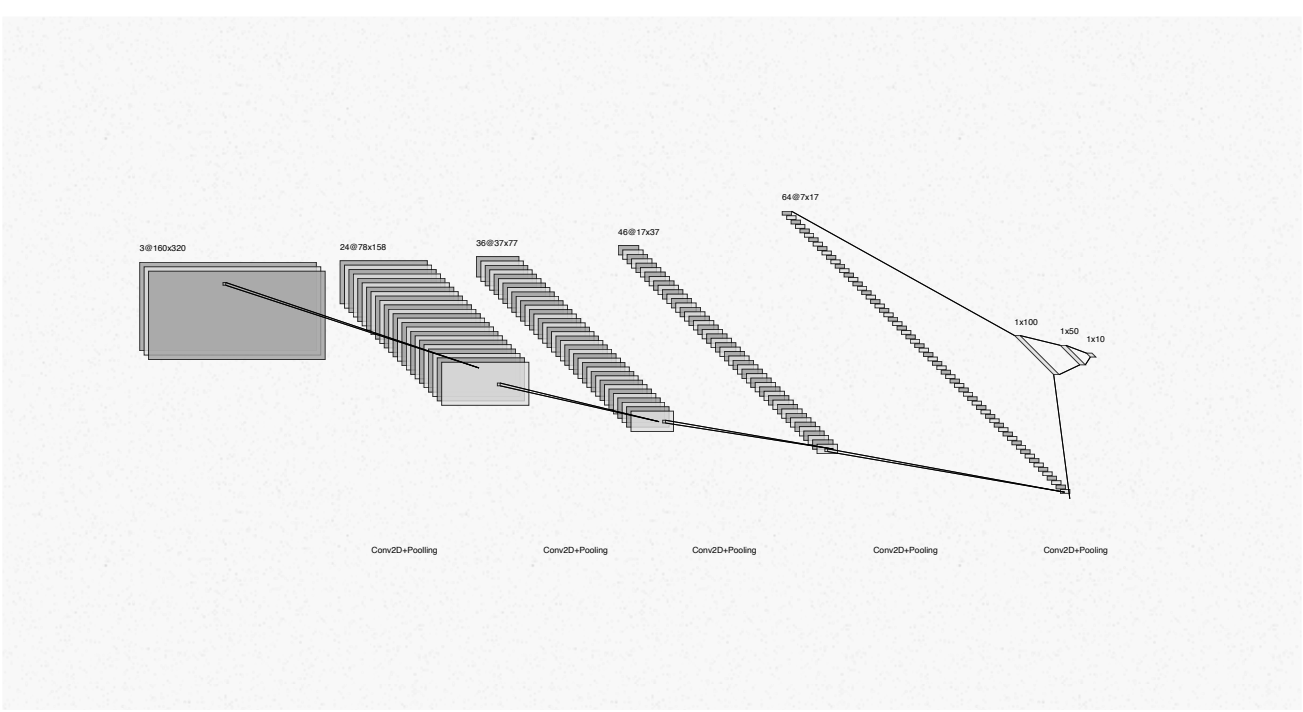
Then I limit the epochs of training to avoid over fittings based on the chart from training result visualization.

The final 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 especially at the right turn after the bridge. To improve the driving behavior in these cases, I record a reverse driving training data to combat the weak point of turning right.

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

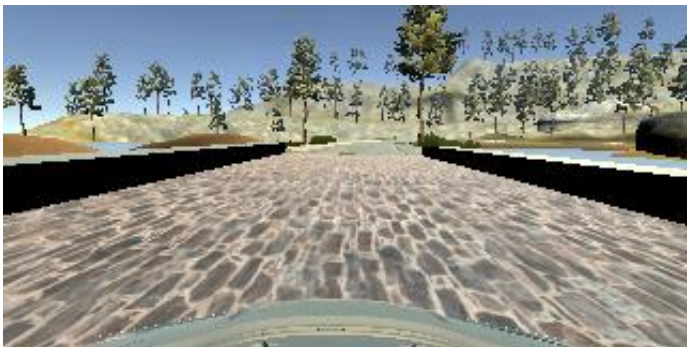
2. Final Model Architecture

The final model architecture (model.py lines 247-260) consisted of a convolution neural network with the following layers and layer sizes as shown in this visualization of the architecture.



3. Creation of the Training Set & Training Process

The interesting part of data collection is most of the data I collected is not used in the final training model. That's because at the early stage of recording a training data, I had a lot of places accidentally drive off the road or making harsh turns. These data harm the training result and make the car drive onto the edge of the road. So at the final stage I try to use the default data only. I find the training result has a problem making harsh right turns. So I use the reverse data as an implement and it worked. The reverse data is made to cover the edge part of the road so that the vehicle can know what to do when it goes to the edge.



After the collection process, I had 11514 number of data points. I then preprocessed this data by flip each image horizontally. I also made use of the side camera images. These procession happened in the generator.

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 8 as evidenced by the flatten of the curve of validation MSE. I used an adam optimizer so that manually training the learning rate wasn't necessary.