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

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

I have used NVIDIA's model architecture as mentioned in the paper "End to End Deep Learning for self Driving Cars". The model consists of 3 Convolutional layers of different depths with kernel size 5x5 and stride 2x2 followed by 2 convolutional layers of kernel size 3x3 and stride 1x1. Every convolutional layer uses ReLU activation. Finally, the model has 3 fully connected layers, where the last one predicts the steering angle.

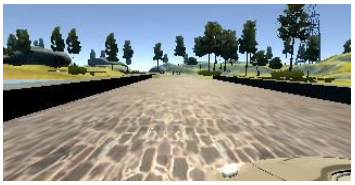
2. Attempts to reduce overfitting in the model

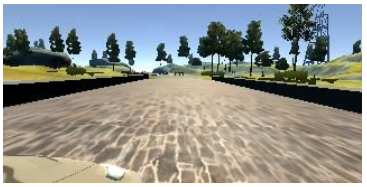
The model contains dropout layers in order to reduce overfitting (model.py lines 51,54). The training data is shuffled for each epoch as the parameter shuffle=True is chosen. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 59).


Center camera


Left camera


Right camera

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. The sample dataset was used for this project.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to avoid overfitting as well s underfitting. Further, the model had to be fast and robust to be feasible for training. Therefore, I used NVIDIA's model architecture and decided to make changes into it as per need. I thought this model might be appropriate because it had already been used for a similar application so it seemed like a good start.

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 introduced dropout layers after the fully connected layers.

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

2. Final Model Architecture

Layer	Description
Input	160x320x3 RGB image
Lambda	Normalisation
Cropping2D	Cropping the image
Conv2D @24	5x5 kernel, 2x2 stride, ReLU activation
Conv2D @36	5x5 kernel, 2x2 stride, ReLU activation
Conv2D @48	5x5 kernel, 2x2 stride, ReLU activation
Conv2D @64	3x3 kernel, 1x1 stride, ReLU activation

Conv2D @64	3x3 kernel, 1x1 stride, ReLU activation
Flatten	Flatten the layer
Fully connected	Output = 100
Dropout	Keep_probability = 0.5
Fully connected	Output = 10
Dropout	Keep_probability = 0.5
Fully connected	Output = 1

3. Creation of the Training Set & Training Process

I have used the sample driving dataset provided by udacity. Before feeding it into the model, I had pre processed the data by normalizing the images. Further, the images were cropped from the top and bottom to remove the car's dashboard and the sky and trees, which were irrelevant to the training of the model.

To augment the dataset, the data was collected by the left, right and center camera and was included in the sample dataset. The dataset was sufficient for the first track, however, to run the car on the second track, more data needs to be collected by driving the car on the second track.

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 5 as the validation loss didn't reduce much beyond that. I used an adam optimizer so that manually training the learning rate wasn't necessary.