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 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 consists of a convolution neural network with 3 convolution layers with 5x5 filter sizes and depths of 16, followed by a fully connected layer of 100 (model.py lines 48-66) The model includes RELU layers to introduce nonlinearity (code lines 48, 52 and 56). Between the convolution layers, there are max pooling layers with pool sizes 2x2.
The data is normalized in the model using a Keras lambda layer (code line 44) which is followed by a cropping layer (line 45).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 58 and 63).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 69). 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 68).

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, using all three cameras and driving on the other direction.

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 to first create a relatively simple network and focus on selecting a representative sample of data and then refine the network by adding more parameters while at the same time control the overfitting.

My first step was to use a simple convolution neural network model with two convolution layers and a fully connected layer. I thought this model might be appropriate for a starting model that would allow me to build on later on.

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 started with only the center data points and it was clear the the model was not able to follow the track since the majority of the steering data was centered around zero, so I added more data that would help in the recovery of the vehicle from the sides. This step was quite helpful but I would still found that the 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. Then I also added a third convolutional layer and that made a significant difference to both the training and validation scores.

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 because it was difficult to distinguish the road and the off-road terrain. To improve the driving behavior in these cases, I recorded a few more samples from these specific areas and also tuned the steering correction parameter, which made a significant difference on the ability to drive around the track.

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 18-24) consisted of a convolution neural network with the following layers and layer sizes:

Layer	Description	# Params
Input	160x320x3 Color image	0

Input	160x320x3 Color image	0
Lambda	Normalization, output 160x320x3	0
Cropping	Output 80x320x3	0
Convolution	5x5 stride, valid padding, output 76x316x16	1216
RELU		0
Max pooling	2x2 stride, output 38x158x16	0
Convolution	5x5 stride, valid padding, output 34x154x16	6416
RELU		0
Max pooling	2x2 stride, output 17x77x16	0
Convolution	5x5 stride, valid padding, output 13x73x16	6416
RELU		
Max pooling	2x2 stride, output 6x36x16	0
Dropout	0.75	0
Fully connected	3456 x 100	345700
Dropout	0.5	0
Fully connected	100 x 1	101

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded one lap on track one using center lane driving using the “normal” direction and another lap going towards the opposite direction. Here is an example image of 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 move towards the center of the road. These images show what a recovery looks like starting from the right side of the road:



To augment the data sat, I also used the images from the left and right cameras, adjusting the correction parameter. The final value for the steering corection was 0.29. I also tried to use flipped images and angles, but it turned out that the reults were better using the left and right cameras, so I removed the flipped images from the training and test sets.

After the collection process, I had 21552 number of data points. I then preprocessed this data using normalization. I also cropped the images by removing 50 pixels on the top part and 30 from the bottom part of each image. Both preprocessing steps were implemented as keras layers.

During model fit I randomly shuffled the data set and put 20% of the data into a validation set. These two settings were part of the keras model.fit() function.

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 in the end was 5. I used an adam optimizer so that manually training the learning rate wasn't necessary.

4. Final thoughts

Although I ended up not using many images from the second track I tried not to collect too many images from the first track because that would essentially mean that the model learns this specific track. As an improvement of the current model I think it would help to make the model better in generalization by utilizing both tracks. That would also require further preprocessing steps when it comes to colour pallets as well as adding a generator step which was not necessary with the amount of data that I used.