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:
* PiloNet_clone_generator.py - containing the script to create and train the model
* drive.py - for driving the car in autonomous mode
* PiloNet_model_5Conv_Generator.h5 - containing a trained convolution neural network
* writeup_report.md & writeup_report.pdf - summarizing the results
* run1.mp4 - for driving video in autonomous mode

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 PiloNet_model_5Conv_Generator.h5
```

3. Submission code is usable and readable

The `PiloNet_clone_generator.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 three 5x5 filter sizes,depths between 24 and 48, two 3x3 filter sizes, depths 64. Then follows four FC layers.
(`PiloNet_clone_generator.py` lines 92-102)

The model includes ReLU layers to introduce nonlinearity almost every layer except Flatten Layer and Dropout Layer, includes tanh layers in final FC layer. and the data is normalized in the model using a Keras lambda layer (code line 89).

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets, add data augmentation and dropput layer to reduce overfitting (code line 44-69 and line 98). 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
(`PiloNet_clone_generator.py` line 104).

4. Appropriate training data

Because the network connection issue, it is very hard to drive car in simulator, so I used sample data provided by Udacity.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was follow the PiloNet introduced by

NVIDIA's paper Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car.

The network consists of 12 layers, including a normalization layer, 5 convolutional layers and 4 fully connected layers. According to the paper, this model can found salient objects on the road such as the edge of the road, to steer car.

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 a dropout layer after first fully connected layer. Just run 3 epochs, the mse is very small, but mean squared error on the training set still higher than validation set.

I want to see how well the car was driving in the simulator. The vehicle seems driving well, and is able to drive autonomously around the track without leaving the road. Thought I think there is a little overfitting.

2. Final Model Architecture

The final model architecture (`PiloNet_clone_generator.py` lines 89-101) consisted of a convolution neural network with the following layers and layer sizes:

- normalization layer
- convolution layer 5x5x24 stride 2x2
- convolution layer 5x5x36 stride 2x2
- convolution layer 5x5x48 stride 2x2
- convolution layer 3x3x64
- convolution layer 3x3x64
- Flatten layer
- FC layer
- Dropout layer p=0.5
- FC layer
- FC layer
- FC layer

3. Creation of the Training Set & Training Process

I augmented only on the training data set, including flipping images and angles, combine with

left and right cameras images with adjusted steering angles. I think it will be combat the overfitting.

After the collection process, I split the data into train set and validation set, 6428 number of training data points, 1608 samples for validation. Then preprocessing the images by normalizing and mean center, and crop the image 70 rows pixels from the top of the image, 25 rows pixels from the bottom of the image.

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 3 as evidenced by the mse is very small, I used an adam optimizer so that manually training the learning rate wasn't necessary.