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](https://review.udacity.com/#!/rubrics/432/view) 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

* video.mp4 with recorded lap

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

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 (model.py lines 100-117, it is nVidia architecture from the lectures):

```
input_shape = (160, 320, 3)
model = Sequential()
model.add(Cropping2D(cropping=((65,30), (0,0)), input_shape=(160,320,3)))
model.add(Lambda(lambda x: x/255.0-0.5))
model.add(Convolution2D(24, 5, 5, subsample=(2,2), activation = "relu"))
model.add(Convolution2D(36, 5, 5, subsample=(2,2), activation = "relu"))
model.add(Dropout(0.2))
model.add(Convolution2D(48, 5, 5, subsample=(2,2), activation = "relu"))
model.add(Convolution2D(64, 3, 3, activation = "relu"))
model.add(Convolution2D(64, 3, 3, activation = "relu"))
model.add(Flatten())
model.add(Dense(100))
model.add(Dropout(0.2))
model.add(Dense(50))
model.add(Dropout(0.2))
model.add(Dense(10))
model.add(Dense(1))
```

The input shape is (160, 320, 3) as the size of the image

The sequential model has layers:

Cropping – to cut unneeded part of images

Lambda – to normalize the image

Convolution with kernel 5, 5 and 24 layers, activation Relu

Convolution with kernel 5, 5 and 36 layers, activation Relu

Convolution with kernel 5, 5 and 48 layers, activation Relu

Convolution with kernel 5, 5 and 64 layers, activation Relu

Convolution with kernel 5, 5 and 64 layers, activation Relu

Next we have flatten and 4 fullyconnected layers with output 100, 50, 10, and finaly 1 value

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 106, 112, 114) with 0.2 drop.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 86, 87). 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 122).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, side cameras image and flipped image of every recorded frame. I've also added bridge data, as the car has a problems with stay on it, because of different texture.

Model Architecture and Training Strategy

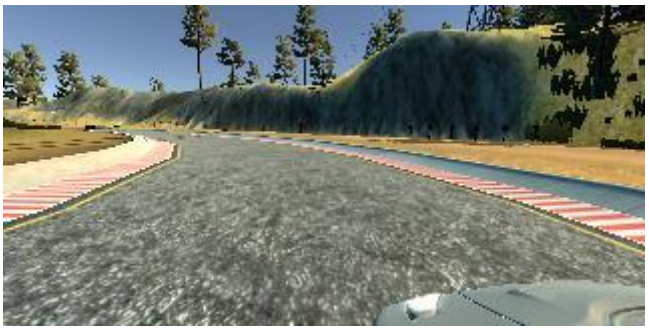
1. Solution Design Approach

The overall strategy for deriving a model architecture was to record data from center lane driving and using correction of 0.2 for the side cameras images.

Data from counter-clockwise track

D:\Marcin\Udacity\SelfDriving cars\CarND-Behavioral-Cloning-P3-master\data1\IMG\center_2017_07_06_23_38_40_680.jpg,
D:\Marcin\Udacity\SelfDriving cars\CarND-Behavioral-Cloning-P3-master\data1\IMG\left_2017_07_06_23_38_40_680.jpg,
D:\Marcin\Udacity\SelfDriving cars\CarND-Behavioral-Cloning-P3-master\data1\IMG\right_2017_07_06_23_38_40_680.jpg,
0,0.4,0,1.868128

Center image with angle 0



Left image with angle correction 0.2

Right image with angle correction -0.2

Data from clockwise track

D:\Marcin\Udacity\SelfDriving cars\CarND-Behavioral-Cloning-P3-master\data2\IMG\center_2017_07_06_23_51_11_605.jpg,
D:\Marcin\Udacity\SelfDriving cars\CarND-Behavioral-Cloning-P3-master\data2\IMG\left_2017_07_06_23_51_11_605.jpg,
D:\Marcin\Udacity\SelfDriving cars\CarND-Behavioral-Cloning-P3-master\data2\IMG\right_2017_07_06_23_51_11_605.jpg,
0.1802575,0.4,0,30.03395

Center with angle 0.1802575

Left with angle 0.1802575 +0.2

Right with angle 0.1802575 -0.2

I used the architecture from the lecture, as the successful model of CNN.

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 had similar test and validation error. But the behaviour of the car wasn't perfect on the bridge. I've added some data of bridge recovery. I've also create 3 clockwise and 3 counter clockwise records of the track.

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

All the recorded images has been flipped to create more training data.

After the collection process, I had 99900 number of data points.

I finally randomly shuffled the data set and put 10% of the data into a validation set. The main test was to drive a vehicle in simulator, so I prefer to have more training data.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The number of epochs was 4, as this was enough to teach the network