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.pdf* summarizing the results

* *video.mp4* demonstrating the capability of the network to control the vehicle during one 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 model.h5*.

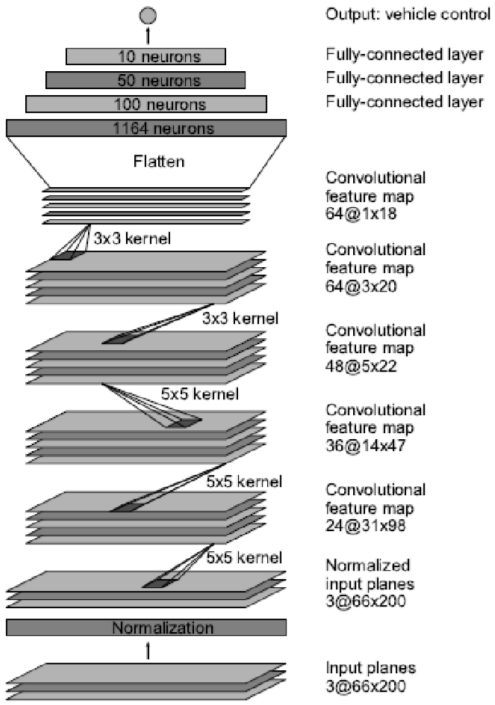
3. Submission code is usable and readable

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

I took as a reference the CNN architecture described in the paper from NVIDIA “End to End Learning for Self-Driving Cars”.



Summarizing, my model starts with a Cropping2D layer (code line 51) in order to remove non-important info that may distract the network beyond the horizon. So, I decided to remove the top 60 lines, and the bottom 30 lines of the captured data.

I considered to remove the most of it to get smaller images and faster training times. If I had no success, I would have tried keeping more lines. Final image size is 320x70.

As suggested in the lessons I included a Lambda layer (code line 52) to normalize the data.

Following, is a series of 5 Convolution2D layers (code line 53-65) with RELU activators. The model has a series of fully connected layers at the end ((code line 68-76)

2. Attempts to reduce overfitting in the model

Initially I didn't add any dropout layers, and the results in the simulator were discouraging. Hence, I tried different configurations. Firstly, I used dropout layers only in the final stages (Fully connected layers) and then just in a couple of Convolutional layers only.

In the end, it turned out that the best results were obtained including a Dropout layer after every Convolutional layer, after Flattening and before the last layer (code lines 54-74).

Other attempts to reduce overfitting were to include training data of the vehicle running clockwise.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (*model.py* line 82). I was also considering to change the Dropout percentage from 0.5 to other value, but it was not finally required.

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 where I saw that the model was having trouble staying inside the lane.

The left and right camera were used with a correction of 0.2. One iteration was made with 0.3 with no noticeable effect.

Model Architecture and Training Strategy

1. Solution Design Approach

The starting point was the NVIDIAs architecture as this configuration has been already tested for a similar purpose. To minimize overfitting the dropout layers were added.

Once the processing and augmentation of the data, along with the architecture of the network is defined, the solution was just a matter of adding additional samples (run minutes). So, I included the line to load previous weights (code line 79) and continue adding additional samples in situations where the model was not preforming correctly.

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 49-76) consisted of

	Layer	Description
1	Cropping2D	Input = 160x320x3 – RGB Image Output = 70x320x3
2	Lambda	Normalization: div by 255 minus 0.5 offset
3	Convolution 1	Kernel 5x5. Stride 2. Padding 'Valid'
4	Relu	Activation
5	Dropout	Keep_prob 0.5
6	Convolution 2	Kernel 5x5. Stride 2. Padding 'Valid'
7	Relu	Activation
8	Dropout	Keep_prob 0.5
9	Convolution 3	Kernel 5x5. Stride 2. Padding 'Valid'
10	Relu	Activation
11	Dropout	Keep_prob 0.5
12	Convolution 4	Kernel 3x3. Stride 1. Padding 'Valid'
13	Relu	Activation
14	Dropout	Keep_prob 0.5
15	Convolution 5	Kernel 3x3. Stride 1. Padding 'Valid'
16	Relu	Activation
17	Dropout	Keep_prob 0.5 18 Flatten
19	Dropout	Keep_prob 0.5
20	Fully Connected	Output = 100.
21	Fully Connected	Output = 50
22	Fully Connected	Output = 10
23	Dropout	Keep_prob 0.5
24	Fully Connected	Output = 1 for the steering angle

3. Creation of the Training Set & Training Process.

To capture good driving behavior, I first recorded four laps on track one using center lane driving. Here is an example image of center lane driving:

I then trained the neural network and studied the weak points where it was prone to leave the road. I noticed that the curves with water in the background (there are 2) were most prone to fail. Also, curves where the side of the road was changing (like the dirt road after the bridge). I decided to add extra data going through these curves up to 4 times per curve. Besides, I recorded the vehicle recovering from those places, before crossing the line to back to center so that the vehicle would learn to how to return to the center. These images show what a recovery looks like:

As previously stated, I used left and right images and then I flipped all the data, along its respective angle measurement.


left view camera


center view camera


right view camera

Shuffle parameter was activated (code line 83 - shuffle=True) in the Keras fit argument to shuffle the data in every epoch.

At the end, the total training data comprised of 72918 samples. 58334 were used for training and 14584 for validation. (20%).

I used 20 epochs. The first epoch lasted 355 seconds, the following epochs lasted more or less a minute each.

```
  model.fit(X_train, y_train, validation_split=0.2, shuffle=True, nb_epoch=20)
Train on 58334 samples, validate on 14584 samples
Epoch 1/20
58334/58334 [==============================] - 355s 6ms/step - loss: 0.0199 - val_loss: 0.0989
Epoch 2/20
58334/58334 [==============================] - 76s 1ms/step - loss: 0.0201 - val_loss: 0.1151
Epoch 3/20
58334/58334 [==============================] - 56s 959us/step - loss: 0.0203 - val_loss: 0.1043
Epoch 4/20
58334/58334 [==============================] - 55s 939us/step - loss: 0.0198 - val_loss: 0.0984
Epoch 5/20
58334/58334 [==============================] - 54s 931us/step - loss: 0.0204 - val_loss: 0.1171
Epoch 6/20
58334/58334 [==============================] - 54s 934us/step - loss: 0.0201 - val_loss: 0.1054
Epoch 7/20
58334/58334 [==============================] - 55s 935us/step - loss: 0.0201 - val_loss: 0.1017
Epoch 8/20
58334/58334 [==============================] - 55s 936us/step - loss: 0.0200 - val_loss: 0.0978
Epoch 9/20
58334/58334 [==============================] - 54s 931us/step - loss: 0.0204 - val_loss: 0.1017
Epoch 10/20
58334/58334 [==============================] - 55s 943us/step - loss: 0.0199 - val_loss: 0.1024
Epoch 11/20
58334/58334 [==============================] - 55s 944us/step - loss: 0.0200 - val_loss: 0.1137
Epoch 12/20
58334/58334 [==============================] - 54s 932us/step - loss: 0.0203 - val_loss: 0.1045
Epoch 13/20
58334/58334 [==============================] - 55s 936us/step - loss: 0.0203 - val_loss: 0.1011
Epoch 14/20
58334/58334 [==============================] - 54s 933us/step - loss: 0.0203 - val_loss: 0.1025
Epoch 15/20
58334/58334 [==============================] - 56s 953us/step - loss: 0.0201 - val_loss: 0.0999
Epoch 16/20
58334/58334 [==============================] - 55s 940us/step - loss: 0.0207 - val_loss: 0.0995
Epoch 17/20
58334/58334 [==============================] - 54s 927us/step - loss: 0.0201 - val_loss: 0.1123
Epoch 18/20
58334/58334 [==============================] - 54s 924us/step - loss: 0.0205 - val_loss: 0.0969
Epoch 19/20
58334/58334 [==============================] - 54s 930us/step - loss: 0.0201 - val_loss: 0.1047
Epoch 20/20
58334/58334 [==============================] - 55s 935us/step - loss: 0.0199 - val_loss: 0.0943
```