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

* run1.mp4 video record of 1 lap autonomous drive

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

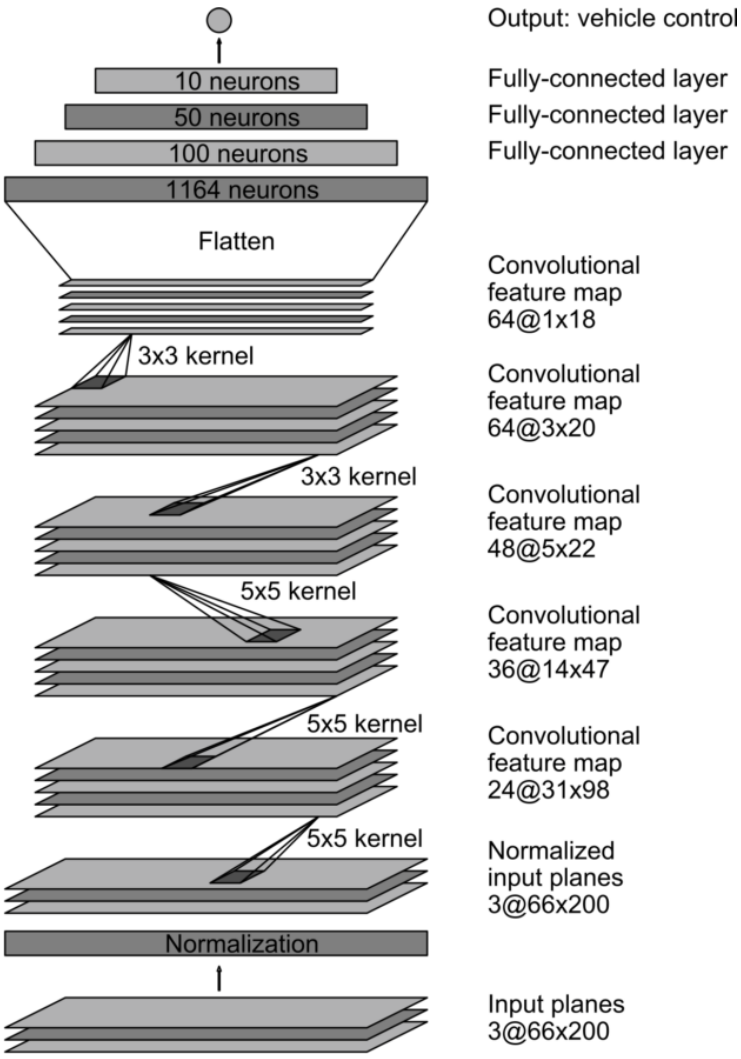
1. Model architecture

I used as base the NVIDIA's CNN (https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/).

1.1 Description of the network:

The first layer of the network performs hard-coded image normalization. It following 3 convolutional layers with strided convolutions with a 2×2 stride and a 5×5 kernel. Next 2 convolutional layers are non-strided with a 3×3 kernel size. The RELU activation is used for each convolutional layer. The five convolutional layers are followed by three fully connected layers, leading to a final output control.

1.2 Visual representation



2. Attempts to reduce overfitting in the model

The model contains GaussianDropout layer at the beginning in order to reduce overfitting (model.py lines 75). Chosen rate is equal to 2 where **rate**: float, drop probability (as with `Dropout`). The multiplicative noise will have standard deviation `sqrt(rate / (1 - rate))`

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 86).

nb_epochs was chosen by several time training the CNN and check the MSE and final behavior.

4. Appropriate training data

It was done several different experiments by driving several laps on the road in both direction (clockwise and counter-clockwise). At the end was chosen to be used only one lap training data but with artificially increased number of data using both center, left and right camera, flipped images for each case and correction factor which was experimentally chosen for smooth turning providing recovery mechanism in case that the car goes at wrong side of the road.

Also for better picture quality was used cv2.GaussianBlur for each image.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

So the model was trained on 10 Epochs by 6259 samples, validate on 1565 samples.

5. Training Strategy and number of Epochs

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 as evidenced by graphical representation of history produced from model.fit() function. I used an adam optimizer so that manually training the learning rate wasn't necessary.

6. Used literature:

https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/

http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

https://keras.io/layers/noise/