Behavioral Cloning: End-to-End Learning

CarND-Term1 Project 3

Author: Bugra Turan, 9.2.2018

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
- **video.mp4** showing the car in auto mode on track 1
- **P3-BugraTuran-Report.md** 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

```
python3 drive.py model.h5
```

Furthermore, the simulator is started with the following settings: 640px x 480px and "fastest" as graphics settings

```
./linux_sim.x86_64
```

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.

Problem approach and pipeline

1. Data collection

Here, I will describe my approach. In a first step data sets were created with the simulator. More specifically the following experiments were conducted:

```
data_fwd
data_rev
data_recov_fwd
data_recov_rev
```

These four sets consist of a recording in "forward" direction of 5 laps. A "reverse" direction of 5 laps. A "recovering" mode in "forward" direction which means that the car was explicitly driven to the lane markings and from there back to the center and so on. Finally, one recording of recovering mode in reverse direction.

In some cases the original data provided by Udacity was used as well. It shall be noted that recordings with the keyboard as steering input lead to very bad results. This is most likely due to the lower resolution compared to the mouse input. It took me quite some time to realize this!

1. Preprocessing

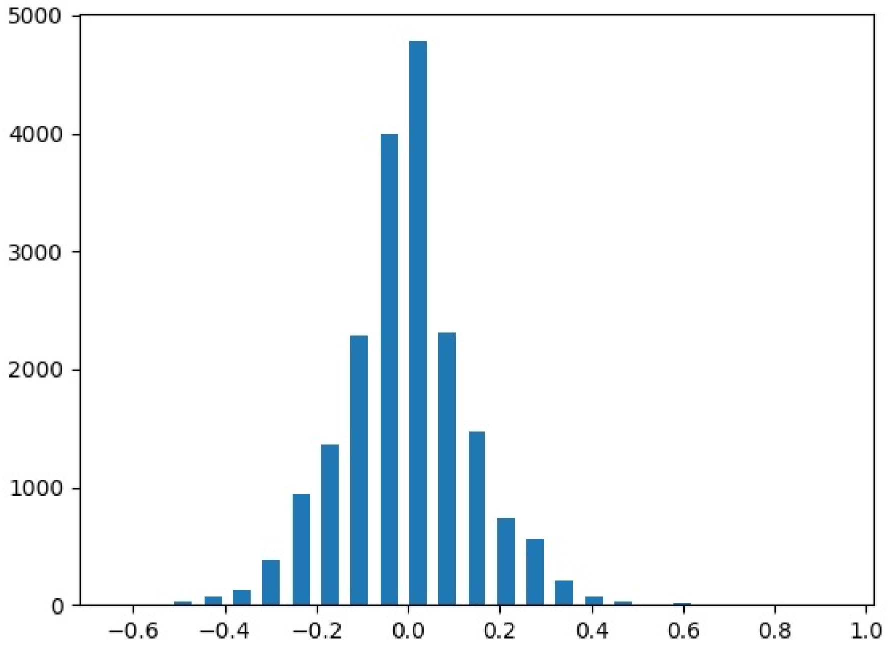
The model.py python script is used for training. There are quite some comments in the script but I will describe the pipeline anyways.

First the script can be started with many parameters as arguments which are parsed on execution. The default values are the ones that were used for the project submission. They are described as the following:

Argument	**Description**	**Default**
'-keep1'	Keep probability of the first Droput Layer	default=0.5
'-keep2'	Keep probability of the second Droput Layer	default=0.25
'-epochs'	Number of epochs	default=5
'-s_div'	Sample divider for training	default=5
'-batch_size'	Batch size	default=128
'-batch_aug_div'	Batch size divider for augmentation	default=16
'-angDep'	Use angle dep augmentation?	default=True
-lr'	Learning rate	default=1.0e-4
'-c_top'	Image crop from top	default=70
'-c_bot'	Image crop from bottom	default=25

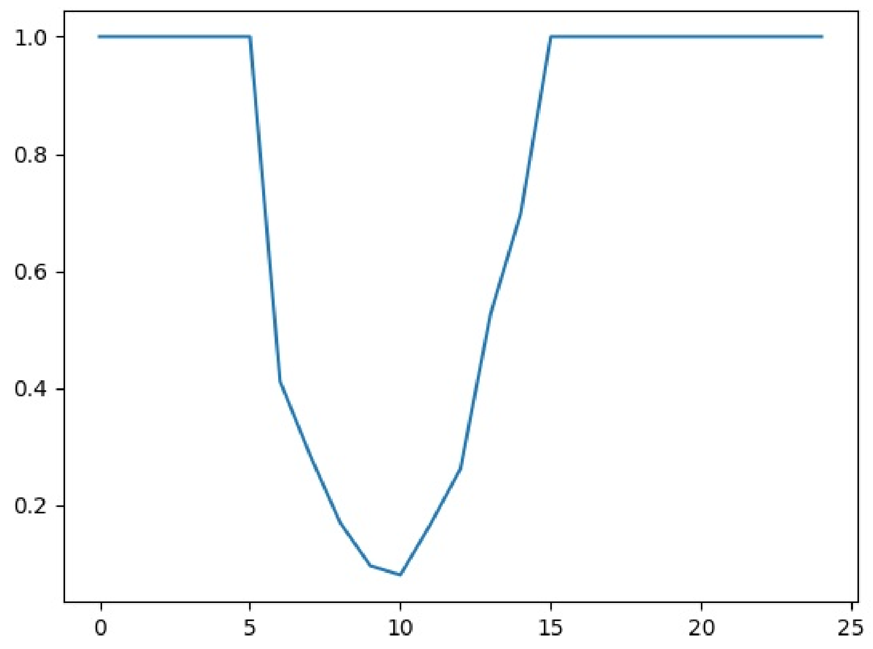
Most of these parameters are self-explanatory except for maybe two. The sample divider for training is used to lower the overall number of train_sample and validation_samples that are being used during training. A large number means that only a small data set is used for training/validation. The second important parameter is the Batch size divider. This value lowers the set set of images/measurements within one set that are candidates for augmentation. The boolean called '-angDep' decides if augmentation is discriminated by the steering angle distribution.

In a next step all the driving logs of the various recordings are being read and shuffled. Afterwards, a histogram is created of the steering angle distribution. Below you can the such a plot.



Number of steering angle occurrences in the overall data set.

It can be seen that obviously the center angle 0° has the highest occurrence. In a next step we will translate this distribution into an inverse which represents the augmentation probability. The function is plotted below for the histogram from above:



Augmentation probability as a function of the steering angle in the overall data set.

We want to augment the angles that are underrepresented. Therefore, the steering angles with a low occurrence have a augmentation probability of 100%.

2. Generators

Now we can split our driving log data into a training and validation set and define the generator functions.
The generator was designed as shown in the Udacity videos. A batch of images and steering angles is loaded. Parts of this batch are candidates for augmentation when the above mentions parameters are set and augmentation probabilities are high. However, it shall be noted that the angle dependent augmentation did not have a huge impact on the results. So it can be turned on or off. When it is turned on a random image/angle is chosen from the set and when the augment probability is high one of four different augmentation operations is applied.

3. Data augmentation

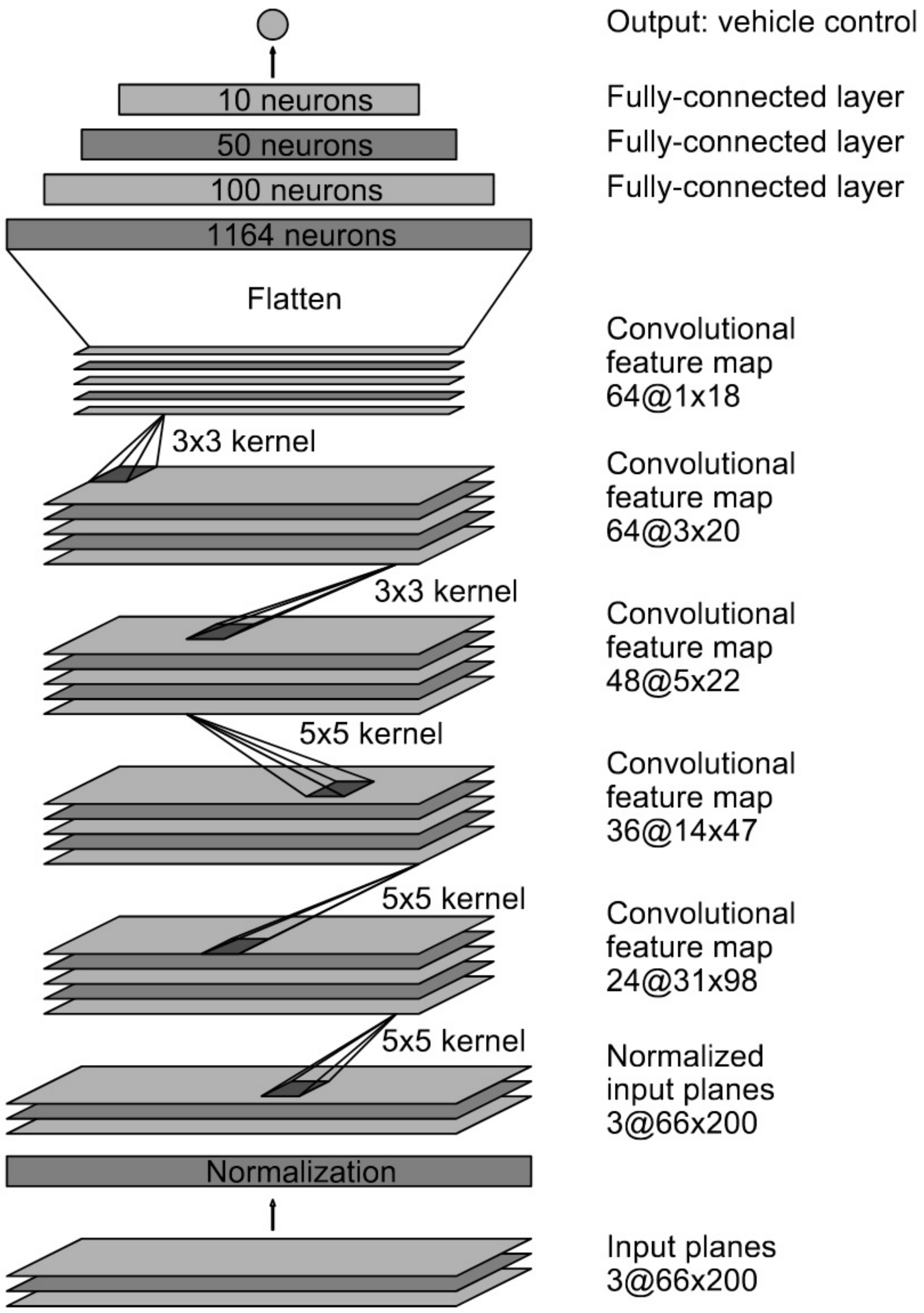
For augmentation either one of four operations is chosen or four different indexes are chosen and all four operations are applied. Both approaches may lead to different outcomes. A further investigation would be helpful. The four operations are:

- Random Shadow
- Random Brightness Adjustment
- Random Vertical Flip
- Random Image Translate

Please note that the data from the other camera positions were not used and only the center camera was of interest.

4. Building the model

The used neural network topology is based on the proposed model from the NVIDIA paper by Bojarski et al. [End to End Learning for Self-Driving Cars]



Proposed neural network. Ref. [1]

However, for better performance two dropout layer were inserted into the network for better stability. All preprocessing on the input is applied within the network graph. This ensures that during driving no extra processing is required in the drive.py. The preprocessing consists of:

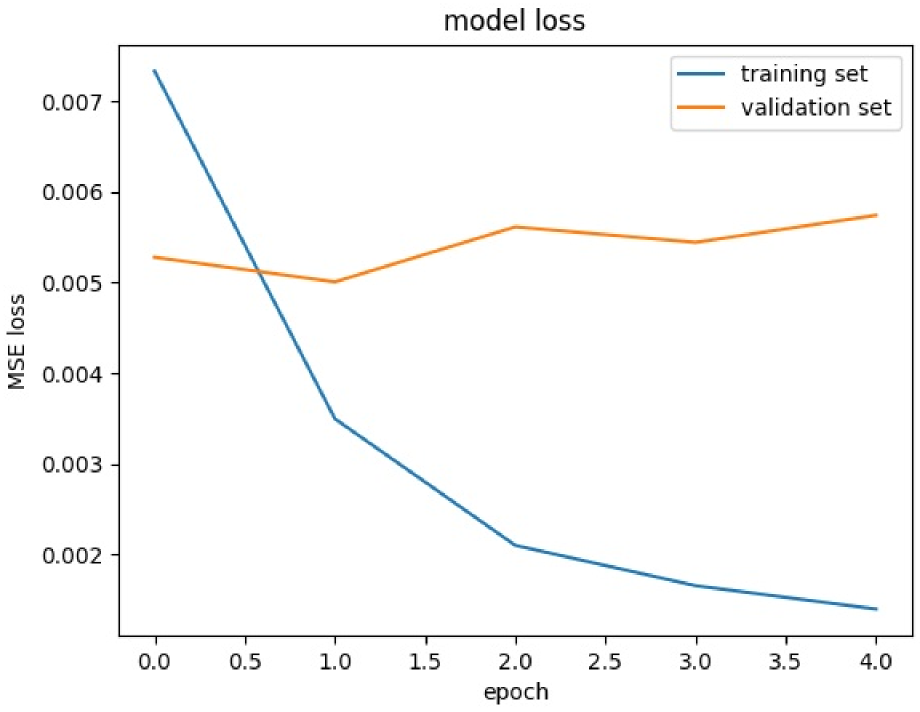
- Image normalization with a Lambda layer applying `X / 127.5 - 1.0`
- Cropping 70 pixel from the top removing the sky and 20 pixel from the bottom to remove the static part of the car.
- Resize the image since in the referenced model an image size of 66px by 200px were used. The Keras backend using Tensorflow is imported in the drive.py as well.

5. Training the model

The model graph is compiled using an Adam optimizer with a learning rate of 0.0001. One particular parameter is defined in the arguments and used for the fit_generator. The value of "samples per epoch divider" adjusts the amount of train_samples and validation_samples that are being used for training. I set a value of 5 which leads to around 3000 samples per epoch. In general not many epochs are needed for a convergence of the validation_losses. Five epochs were used for my submission.

6. Evaluation

The training process is summarized in a results plot. An example is shown below.



Visualization of the training results.

The training loss is reducing continuously while the validation loss stays more or less the same. For more epochs it starts to rise which. Maybe a smaller "samples per epoch" value together with a higher number of epochs could be tested. A smaller batch size (128 in this case) could be of interest as well.

A video of a working prototype using my approach is attached to this submission. Furthermore, I have uploaded a more clear video onto youtube which can be found at:

https://www.youtube.com/watch?v=T3bZXXNaBUw&feature=youtu.be

Using the angle dependent augmentation shows similar performance but with slightly more oscillations in the beginning.

Outlook

As a summary I have to say that I spent the most time for this project up until now. I was recording my data with using the keyboard as input which lead to very bad results and hindered my progress. Changing the color space of the images to YUV in both cases (model.py AND drive.py) seems to have an impact as well? This was mentioned by many students that have done this project before. Finally, reducing the learning rate and adding the two dropout layer had a stabilizing effect as well.

All in all data was the big key in this project. Massively increasing the training set (5 laps each in both directions and recovering maneuvers) had the biggest impact on the results. Much more than any changes to the hyperparameters.

For the future a proper integration of the othe two cameras should be tried out as well as training on the second track.