writeup

April 8, 2020

1 Behavioral Cloning

1.1 Writeup

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

1.2 Rubric Points

1.2.1 Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

1.2.2 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

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
```

The simulator ran in the modus fastest!

In the submitted version of drive.py I removed the gpu functionality as on my local machine this led to some error's. However training worked using the gpu.

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. However, I wrote before in a jupyter notebook ('model.ipynb'), which might be better readable. If it wasn't for the submission i would have split up the 'model.py' file in multiple, one for import, preprocessing, training ...

1.2.3 Model Architecture and Training Strategy

1. An appropriate model architecture has been employed My model is with slight changes in the input layer exactly the model developed by NVIDIA for deep drive functionality: Autopilot CovNet. As this model already proved to be suitable for such a task I also relied on it (and was not disappointed at all).

The model ('model.py', lines: 180-240) is structured as follows:

Layer (type)	Output Shape	Param
batch_normalization (axis=3)	(None, 64, 128, 3)	12
conv2d_0 (Conv2D)	(None, 30, 62, 24)	1824
conv2d_1 (Conv2D)	(None, 13, 29, 36)	21636
conv2d_2 (Conv2D)	(None, 5, 13, 48)	43248
conv2d_3 (Conv2D)	(None, 3, 11, 64)	27712
conv2d_4 (Conv2D)	(None, 1, 9, 64)	36928
flatten_0 (Flatten)	(None, 576)	0
dense_0 (Dense)	(None, 1164)	671628
dense_1 (Dense)	(None, 100)	116500
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

Total params: 925,059
Trainable params: 925,053
Non-trainable params: 6

2. Attempts to reduce overfitting in the model Main measure to reduce overfitting was to shrink the actual input image. The original image has a size of 320x160. Most of it's content does not even cover the road. That's why I removed 65 pixels from the top (mainly heaven and trees) and 20 from the bottom (mainly ego-vehicle) of the image. This gave an image size of 320x75 (e.g. in 'model.py' line 66) In a second step the resolution is reduced to 128x64, given in 'model.py' line 67. These methods are applied directly in the input processing part of my script. That's why I applied the same transformation in 'drive.py' again.

By this measure the size of the neural network was reduced from 32,213,379 to 925,059 parameters of the neural network, leading to the network as shown in the previous section. As an image has 8192 pixels just 113 (independently distributed) images would be - theoretically - sufficient to determine all parameter. That number is about 6 times lower than for the high resultion image. As this seams to be a reasonable low number I continue from here.

3. Model parameter tuning In case of the hyperparameters of the neural network, i also relied on the values given by nvidia. However, i increased the validation size to 25% and reduced the number of epochs to 10, as this value proved to be enough.

4. Appropriate training data Training data was produced by driving the track manually with the keyboard. I tried to keep the vehicle in the center of the track, however this proved to be the hardest challenge of this exercise. I drove the track in a continous run for 5 laps in each direction.

For details about how I created the training data, see the next section.

1.2.4 Model Architecture and Training Strategy

1. Solution Design Approach The overall strategy for deriving a model architecture was to aquire a model this already proved to be sufficient for the deep driving approach.

My first step was to use a convolution neural network model identical to the Nvidias Autopilot CovNet, as it proved reliable results even in real vehicles. However, I reduced the size of the input images for this network further to 128x64 from the original 128x128 size. As most parts of the vertical axis did not include the road this seams to be reasonable.

Presuming to have found a suitable network model I wrote the data import and data processing. During the data import I already shrinked the image data to 128x64, given in 'model.py' lines 66-67. After storing all data - individually for each run - in the list input_data, I saved it to a pickle file and continued with the preprocessing.

The preprocessing mostly covers smoothing and adding additional data. As I drove with a keyboard the steering inputs are far away from smooth. That's why I used scipy's filtfilt method which led to smooth steering signals without a loss in the phase plane. Additionally the images from left and right side are added to the training data with applying an steering offset of 0.3 (left) and -0.3 (right) to the data. (as given in the lesson)

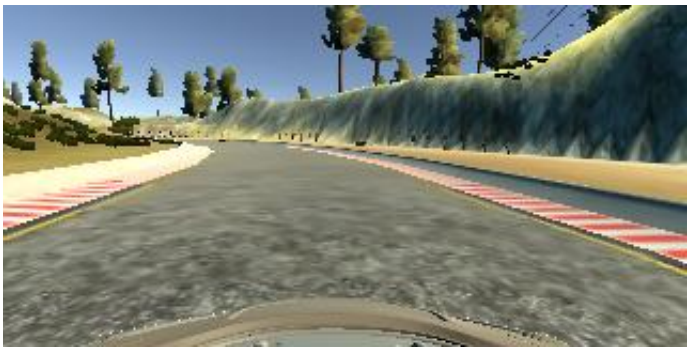
Then I recorded data by trying to drive in the lane center using the keyboard. I recorded two runs one in each direction (clock-wise, counter-clockwise). Each of the runs includes exactly 5 laps.

Using this data I trained my neural network. As it turns out it was very beneficial to record much data from the start as the vehicle drove without any more iterations in the lane center. However, I probably would not accept such a vehicle behavior as it is slightly unsteady among the lane center at straights.

While writing this report the simulation run continously and the vehicle never left the road.

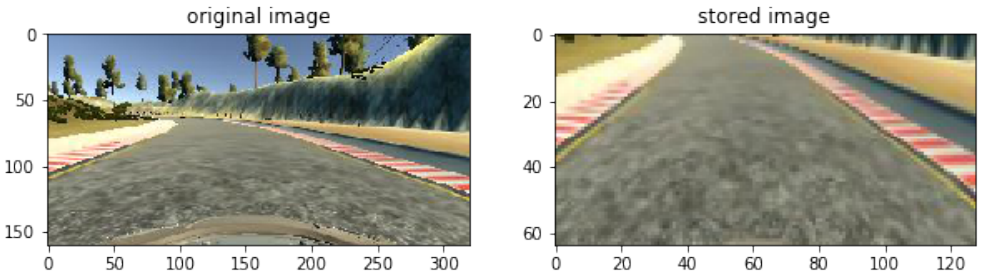
2. Final Model Architecture The final model architcture has been described before.

3. Creation of the Training Set & Training Process To capture good driving behavior, I first recorded five laps on track one trying to drive in the center of the track. I did this once in clockwise and in counter-clockwise track direction. Here is an example image of center lane driving:



alt text

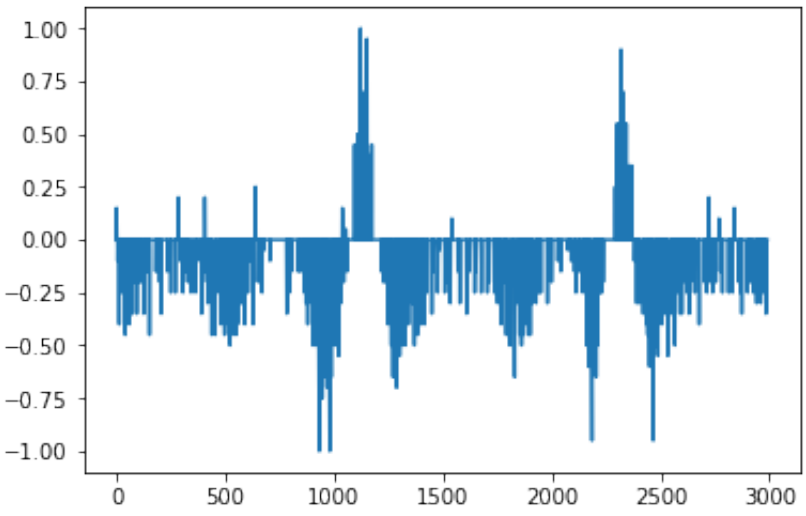
Then I reduced the image size to a useful range of interest, as visible here:



alt text

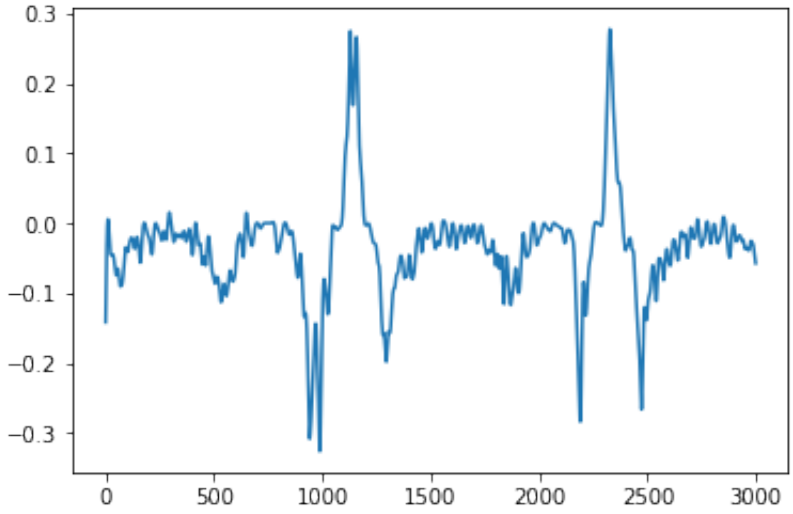
After the collection process, I had 9839 data points, which consists of 3 images (left, center, right) and 4 measurements (steering,throttle,unknown,speed).

Due to the steering based control the steering inputs were very jerky. Here is an example for just one lap:



alt text

By using a phase-free filter (filtfilt + butterworth (order=2,freq=0.5Hz)) these inputs are smoothed before importing them in the neural network:



alt text

Then I merged the images of left and right side in one list for all drives to obtain the final training input for the neural network. However, a steering offset of 0.3 (left) and -0.3 (right) is applied to the reference output of the network (steering angle). This measure accounts for a lane-centering-controller-like behavior.

Additionally the first and last 20 datapoints of each run are removed after filtering, as in random situations the initialization of the phase-free filter lacks in such situations.

I finally randomly shuffled the data set and put 0.25% of the data into a validation set, which was done directly within the keras optimizer.

[ ]: