CarND-Behavioral-Cloning

Udacity Self Driving Car - 3rd Project - Behavioral Cloning

General overview

In the third project of the Udacity Self Driving Car Nanodegree program we should create a CNN in order to steer a car around a pre-defined road.

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
- Readme summarizing the results

Submssion 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

Submssion 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.

Description of data processing

Data generation / training the network

For the project I generated two types of data sets: a training data set and a validation data set. The test data set consists of a real drive in the simulator.

In order to generate the training data I drove four times in a normal way in the circuit. After the first training sessions especially at one point the car always turned tight instead of turning left.

Here are some images of that point:










Hence in order to compensate for that I enriched the data set by focussing on data within the curves. So I recorded my driving in the curves up to 5 times. Moreover, I drove two rounds backwards. Moreover, I introduced random flipping of images in the data process in order to have a good distribution of right/ left curves in the data (and the corresponding steering angles).

After the training the car could drive the curve alone. Here are some images from that position:







This improved the driving behaviour so that at the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

For the validation data set I drove 6 rounds in the circuit and used this data as validation data.

To sum up:

- The training data set comprises approx. 70.000 images (sum of images from center, right, left camera).
- The training data set comprises approx. 10.000 images

Data processing of training data

From the data there is three different kind of data available: images from the right, center or left camera. Following the approach a Nvidia team, an image was randomly chosen. In order to cope with the different right/ left perspectives an additional angle was added the steering angle. After the selection, the following steps were applied to each image:

- Normalisation: A conversion to HSV is used to ensure that the image have the spectrum. HSV is less sensitive with respect to different lights
- Shifted images: In order to get more augmented data the images were randomly shifted either to the left or right side. The steering angle was adapted respectively.
- Cropped image: The image was cropped in order to remove the noise from the background. By cropping the image we assume that each camera has a fixed position and a fixed perspective.
- Flipping of the image: In order to get a good distribution of the steering angle the images were randomly flipped and the steering angle was adapted accordingly.

Data processing of validation data

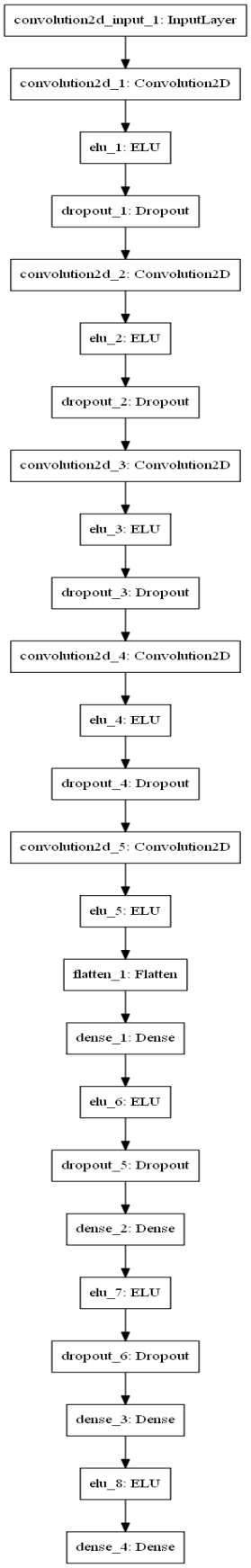
For the validation set only images from the center camera are used. After the selection, the following steps were applied to each image:

- Normalisation to HSV
- Cropping of image
- Randomly flipping of image

Description and Running of CNN

For the CNN I applied 'Transfer Learning' and implemted the CNN from Nvidia. The CNN is described in the table below.

In order to avoid to run out memory the Keras ImageGenerator was used which passed the data in batch sizes from n=128 to the CNN. That data contains a large amount of '0' steering angles. In order to cope with that images with a steeting angles between -.1 and .1 were randomly removed from the data. The model used an adam optimizer, so the learning rate was not tuned manually.



- Convolutional Layer 1: As data input I used images of size 66x200 @ 3 (3 dimensions as images are in HSV format). At this pictures I applied a 2x2 stride and a 5x5 kernel. As activation function ELU function is used (https://arxiv.org/pdf/1511.07289v1.pdf). In order to avoid overfitting a drop-out layer with a drop-out rate of 0.3 is set. The output format is 31x98 @ 24.

- Convolutional Layer 2/ Layer 3: In these layers I applied a 2x2 stride and a 5x5 kernel. As activation function ELU function is used. In order to avoid overfitting a drop-out layer with a drop-out rate of 0.3 is set. In Layer 2 the output format is 14x47 @ 36, for Layer 3 it is 5x22 @ 48.

- Convolutional Layer 4/ Layer 5: In these layers I applied a non-strided convolution with a 3x3. As activation function ELU function is used. In Layer 4 in order to avoid overfitting a drop-out layer with a drop-out rate of 0.3 is set. In Layer 4 the output format is 3x20 @ 64, for Layer 5 it is 1x18 @ 64.

- Flatten 1: For connecting the data to a fully connected network, the data has been flattened (which results in 1154 neurons).

- Dense 1 / 2 / 3 / 4: In these layers the neurons size decreased in order to get a single value as steering angle. Similar to the other layers, as activation function ELU is chosen and in order to avoid overfitting a drop-out rate is set. The layer Dense 1 has 100 neurons, Dense 2 50 neurons, Dense 3 10 neurons, Dense 4 (as final layer) 1 neuron.