Behavioral Cloning

Model Architecture and Training Strategy

1. The used CNN based on NVIDIA model

My model consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 123-172). The maxpooling size is from (3,3) to (1,1). The first stride (3x3) is used to come close to Nvidia output, after the first maxpooling layer.

- ✗ Dave-2: 66x200 stride (2x2) => 31x98
- ✗ model.py: 100x320 stride (3x3) => 32x106

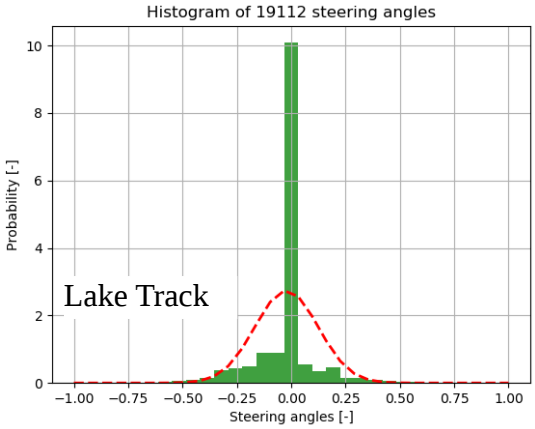
My CNN result in a slightly bigger CNN than Nvidias Dave-2. The model has ten layers. The five 2D Convolution layer, the Flatten Layer and three Dense layer plus the output Dense layer.

2. Attempts to reduce over-fitting in the model

The model contains the “L2 regularizer”, the “batch normalization” and the “dropout” layers (line 129-170). Only the Flatten layer got no “L2 regularizer”. For my understanding it was not clear, why this can not be applied in Keras like in a dens layer.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line174).The “correction” angle for left and right camera was tuned manually to balance between "nerves drunken" driving modus with hight correction and a "straight sleeping" driving modus with low correction. The correction angle was chosen as 0.4 for my results (line 206).



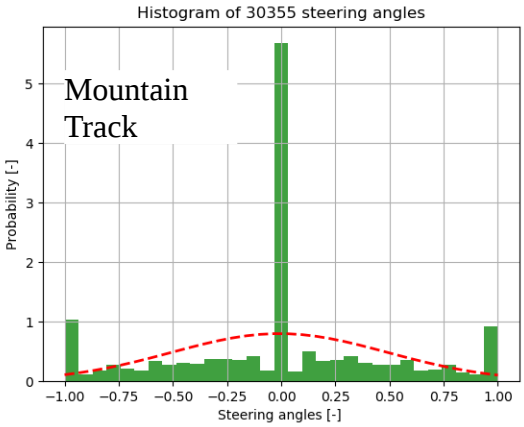
4. Appropriate training data

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 as a starting point. The statistic of the steering angle from the lake shows a maximum at zero degree. Also there is more negative steering to see. Therefor flipping was used (line 101, 106) to keep the car in the middle of the road.

In the mountain are also a maximum at zero degree, but sigma is higher here compare to the lake-data. In the mountain-data is peek at both maximal steering angle. This tells us that there are some sharp curves in the track.

In oder to reduce the zero steering angle, it is recommend to drive often small s-curves. This also enables to learn recovering to center of the road.

But in general driving is mainly a straight ahead thing with some very sharp curves.
So I take the this data without filtering for the training



All together I collect 50 000 sample which are 300 000 images for the CNN to learn driving.

Model Architecture and Training Strategy

1. Solution Design Approach

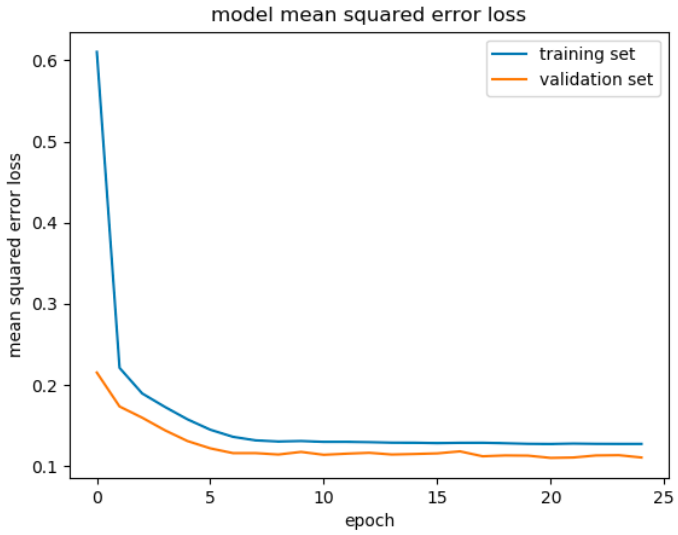
The overall strategy for deriving a model architecture was:

If you want more quality in driving, use much more data.

My first step was to use a convolution neural network model similar to the LeNet. I thought this model might be appropriate, because it small fast. But than i see that i can use a bigger CNN and i take the Nvidia dave-2 architecture as a basis than, because the computer was able to do that.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was over-fitting.

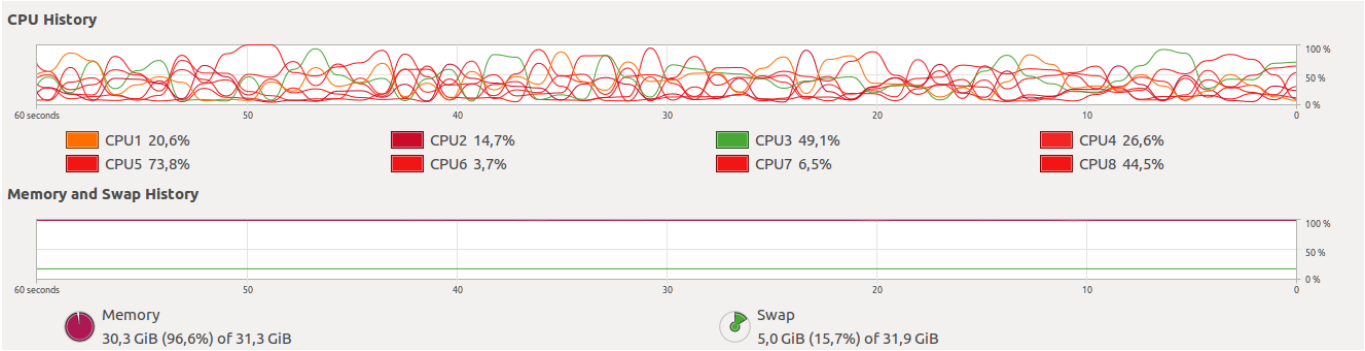
To combat the over-fitting, i implement the L2 kernel_regularizer, batchnormalization and dropout, so that validation loss is lower than the train loss, like shown beside.



The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track, the curve after the bridge. To improve the driving behavior in these cases, I record additional data.
Only with good smooth date i could improve the quality of the critical curves.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road around the lake.

Here you can see a screenshot of my computer during the Training of the 100 000 images from the lake. The Memory is nearly fully used and also 5GiB is swapping on the SSD. The performance goes down a bit. This is the limit of images for taining without the use of the generator.



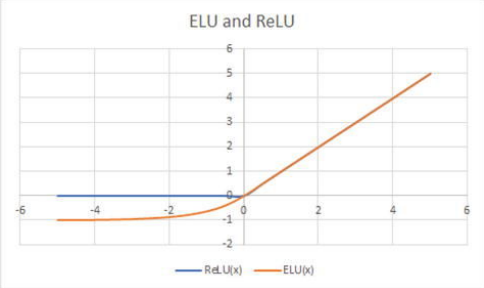
This is the point, where I implemented the generator (model.py line 72-110). The generator needs a “**batch_size”**. I choose the maximum how is possible for my GPU-RAM. That are 32 images.

The difficult on the challenge track is the start area and the two sharp curves. I used the same strategy than on the lake track: six tracks driving in the center of the road with recovering S-curves and smooth curves. In additional I collect data in the difficult areas as needed. In the mountains I **reduce the crop size** of the **input images** from 68x320 to **100x320**. This allowed the CNN to see the hight red signs.

2 Final Model Architecture

The final model architecture (model.py lines 123-172) consisted of a convolution neural network with the following layers and layer sizes:

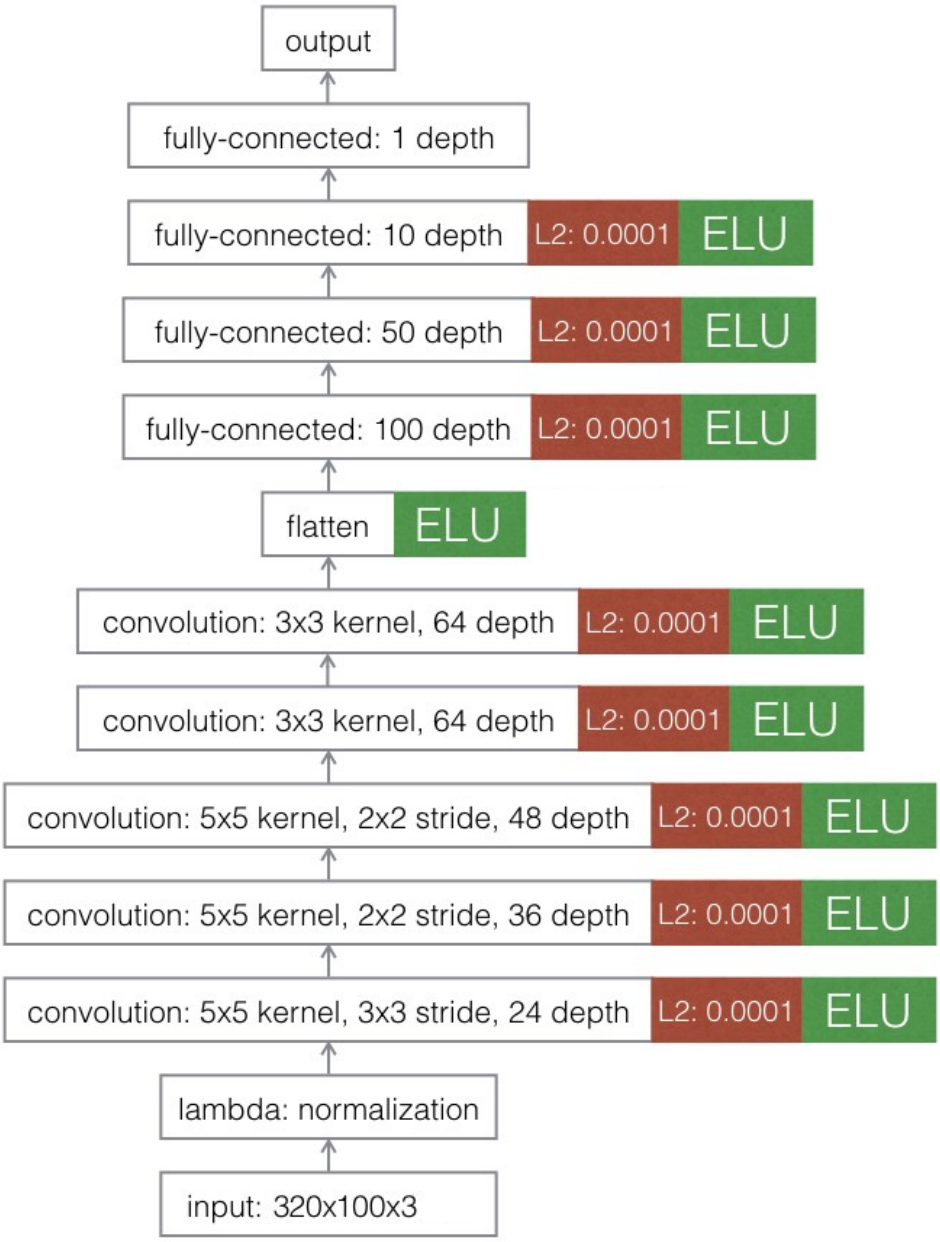
cropping from 160x320x3 to => 3@100x320
Convolution 5x5 and Maxpooling **3x3** => 24@32x106
Convolution 5x5 and Maxpooling 2x2 => 32@14x51
Convolution 5x5 and Maxpooling 2x2 => 48@5x24
Convolution 3x3 and Maxpooling 1x1 => 64@3x22
Convolution 3x3 and Maxpooling 1x1 => 64@1x20
Flatten 100 => 50 => 10 => 1 float output 1@1x1



This close to the NVIDIA Dave-2. The main diffent is the strides=(3,3) line 129 which allows me to reduce the bigger input of my CNN 100*320 to the given 66*200 in an easy way.

For this CNN I used the activation ELU instead of a simple RELU, because its more nonlinear.

The L2 regularizer parameter was set to 0.0001, to minimize the prediction error.



3. Creation of the Training Set & Training Process

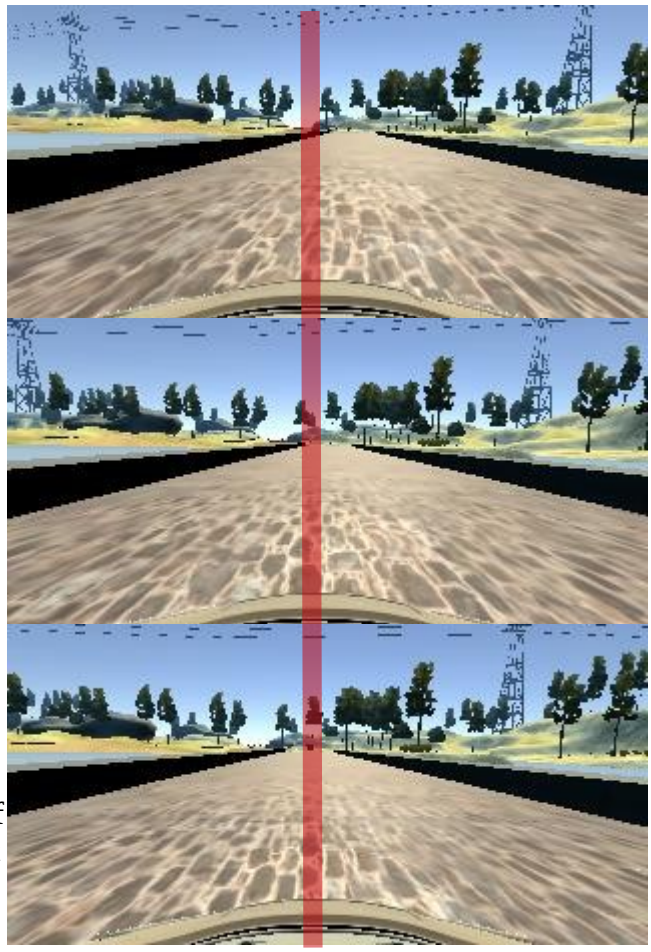
To capture good driving behavior, I first recorded four laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to find the center back. These images show what a recovery looks like starting from the right image:

As you can see with the red line the car will drive to the boundary of the bridge.

The second 5 frames later is recovering.

And the third 5 frames later has already fully recovered and is strait on the way to "the well know curve of P3".



To recover again steer more right and recover again. Then I repeated this process on track two in order to learn the model recover to the middle of the road. Without recovering the car drives straight to boundary of the bride, because the CNN didn't learn to steer on the bridge.

To augment the data sat, I also flipped images and angles thinking that this symmetry would help to drive in the middle of the road. Other augmented images was not generated, because it is so easy to get new images from the simulator. For example, below there is an image that has then been flipped right side:

If the car leave the road. I start collecting images around this area only. This was the case in the curve after the bridge for the lake track, like shown above - "the well know curve of P3".

In the mountains i have problems withe the two sharp curve and add more input images from here. To challenge that i extend the area for the Input from 68 to 100 lines in the middle of the window.
The green lines in the image below shows the cropping frame:

This allows me to feed the CNN with the data of the red signs. I think this helps not to drive straight between the signs and falling down the hill.

After the collection process and flipping, I had 255306*320*68*3= 24 509 376 000 number of data points for the lake and mountains track.

First the data must be preprocessed by converting the color space from RGB to BGR (code line 66).

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

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 10 as evidenced by the calculation time of one night. Better result are possible with more epochs, because the mean squared error loss is going slightly down, but the quality came with the amount of data, not with the number of epochs.

Concusion:

I learn many thinks and im happy with this results.

It is amazing how the input controls the output: Both the Amount and The Quality of Data.

Keras is very good compare with pure TF.

The generator was relay useful with big data sets.

Finlay I like to say that this Project is really fun and I like to do some advanced thinks on it like reinforced learning or the MPC Control, but time is over and this I hope we learn this later - thanks to David Silver.