Use-Deep-Learning-to-Clone-Driving-Behavior

The goals / steps of this project are the following:

- Use the simulator to collect driving pattern of a good driving behavior.
- Visualize the data set and add augmentation
- 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

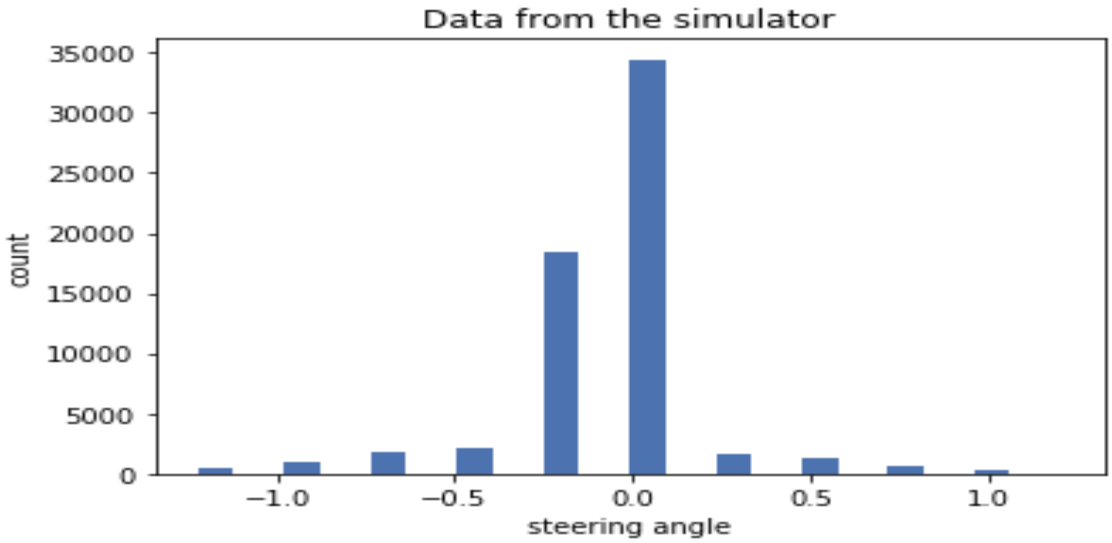
Final Result of the Car successfully inside the track

Creation of the Training Set & Training Process

To capture good driving behavior, I used the data provided by Udacity - about 1 laps of center lane driving and 1 lap of opposite direction and left to center and right to center driving. Here is an example image of center lane driving, left to center driving and right to center driving.

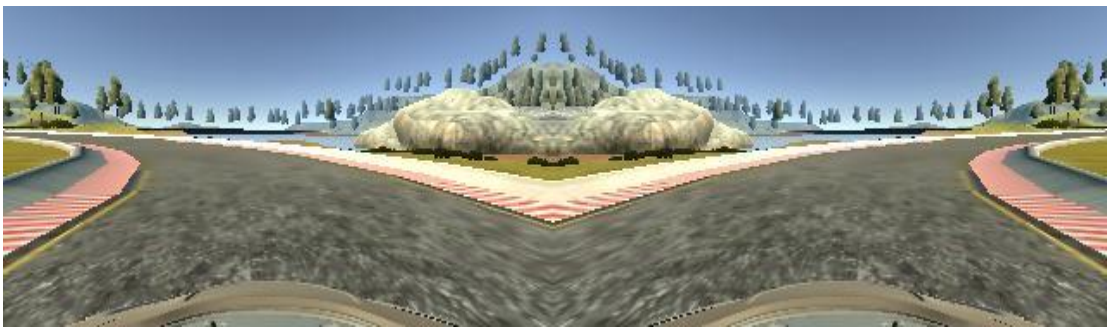
After collecting the above image, I choose steering angles for center lane driving from the recorded data. For left camera view driving, I calculated steering angle as center lane driving + 0.2 and for right camera view driving, steering angle as center lane driving – 0.22.

```
Total data points from the training data is 61866
```

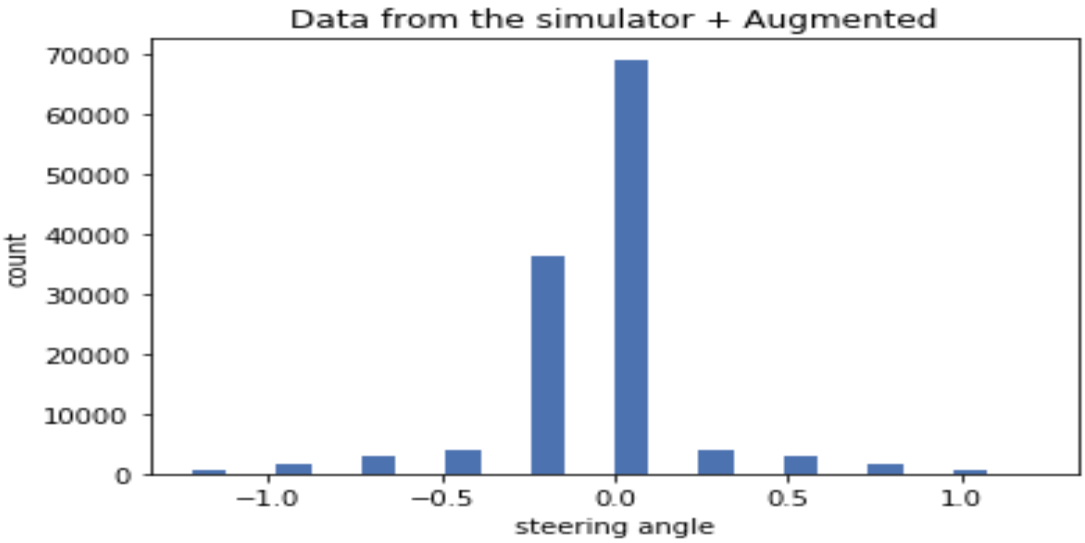


Augmentation

I took 100% of collected simulated data and augmented it. To augment the data set, I flipped images horizontally and multiplied all steering angles with “-1”



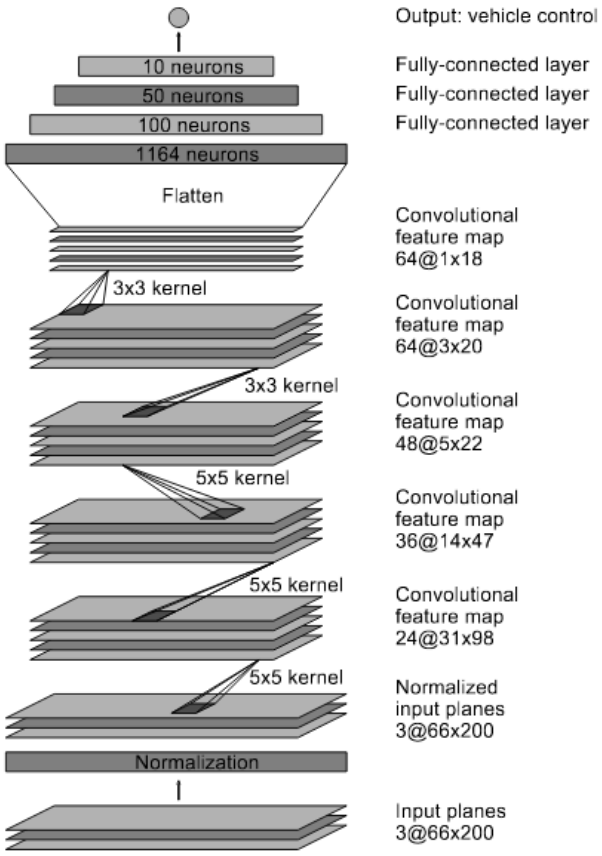
`Total data points from the training data + Augment data is 123732`



Model Architecture and Training Strategy

Throughout the project I tested 3 models: LeNet, model proposed by Comma.ai and model reported in Nvidia paper for end-to-end learning for self-driving cars. The last, Nvidia model turned out to be the best for the data used in the project. It's a well-known Convolutional Neural Network recently, it has about 27 million connections and 250 thousand parameters.

The overall strategy for deriving a model architecture was to try different known models and add/remove layers based on a feeling that given model can generalize well. My first step was to use a convolution neural network model similar to the LeNet model. I thought this model might be appropriate because it worked well in previous project. However, after some investigation I observed that the Nvidia model work better in general.



Attempts to reduce overfitting in the model

The model contains one dropout layers after the flatter layer in order to reduce overfitting (model.py lines 216).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 206). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 234). I tuned a bit the batch size and stayed at batch size equal 32. Also, I was choosing the number of epochs mainly between 5 to 7. Finally, it was sufficient to run the training only on 5 epochs.

Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving and recovering from the left and right sides of the road. For the center lane driving I used the dataset prepared by Udacity - about 8 laps. Then, by driving the simulator

manually, I collected samples for recovery situations. One lap where a car has to recover from different places at the right side of the road, another lap for the left lane recovery. Additionally, I saved one lap of further center lane driving for turn no. 2 and no. 3 as the model had some problems with these places before.

Final Model Architecture

- Image cropping – using Keras Cropping2D. Input image size is 160,320,3 and output image size is 70,320,3
- Image normalization - using Keras lambda layer
- Convolution: 5x5, filter: 24, strides: 2x2, activation: RELU
- Convolution: 5x5, filter: 36, strides: 2x2, activation: RELU
- Convolution: 5x5, filter: 48, strides: 2x2, activation: RELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: RELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: RELU
- Flattening
- Dropout (0.5)
- Fully connected: neurons: 1162, activation: RELU
- Fully connected: neurons: 100, activation: RELU
- Fully connected: neurons: 50, activation: RELU
- Fully connected: neurons: 1 (output), activation: LINEAR

Table

_____

Layer (type) Output Shape Param #

==========

cropping2d_1 (Cropping2D) (None, 70, 320, 3) 0

_____

lambda_1 (Lambda) (None, 70, 320, 3) 0

_____

conv2d_1 (Conv2D) (None, 33, 158, 24) 1824

_____

conv2d_2 (Conv2D) (None, 15, 77, 36) 21636

_____

conv2d_3 (Conv2D) (None, 6, 37, 48) 43248

_____

conv2d_4 (Conv2D) (None, 4, 35, 64) 27712

_____

conv2d_5 (Conv2D) (None, 2, 33, 64) 36928

_____

flatten_1 (Flatten) (None, 4224) 0

_____

dropout_1 (Dropout) (None, 4224) 0

_____

dense_1 (Dense) (None, 1162) 4909450

_____

dense_2 (Dense) (None, 100) 116300

_____

dense_3 (Dense) (None, 50) 5050

_____

dense_4 (Dense) (None, 10) 510

_____

dense_5 (Dense) (None, 1) 11

==========

Total params: 5,162,669

Trainable params: 5,162,669

Non-trainable params: 0

_____

Following are the Epoch wise log

Epoch 00001: loss improved from inf to 0.05558, saving model to model.h5

Epoch 2/5

516/516 [==========] - 143s 276ms/step - loss: 0.0483 - val_loss: 0.0488

Epoch 00002: loss improved from 0.05558 to 0.04828, saving model to model.h5

Epoch 3/5

516/516 [==============================] - 142s 276ms/step - loss: 0.0425 - val_loss: 0.0410

Epoch 00003: loss improved from 0.04828 to 0.04247, saving model to model.h5

Epoch 4/5

516/516 [==============================] - 142s 276ms/step - loss: 0.0388 - val_loss: 0.0400

Epoch 00004: loss improved from 0.04247 to 0.03885, saving model to model.h5

Epoch 5/5

516/516 [==============================] - 143s 276ms/step - loss: 0.0368 - val_loss: 0.0393

Epoch 00005: loss improved from 0.03885 to 0.03676, saving model to model.h5

The final video is archived in the github repo