Project 3. Behavioral Cloning.
Writeup.

Goal of the project: Build a Convolutional Neural Network for predicting the steering angles for controlling the car on simulated race track in autonomous mode.

The steps of this project are the following:

- Collect the data for training the neural network
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with collected training and validation datasets
- Test that the model successfully drives around track 1 without leaving the road
- Summarize the results

Project resources

My project includes the following required files:

- **model.py** contains the script to create and train the model
- **drive.py** script for driving the car in autonomous mode using trained model
- **model.h5** containing a trained convolution neural network
- **writeup.pdf** summarizing report of the work done in scope of this project
- **video.mp4** video recording of simulated car controlled by trained model on track 1 at speed 15mph
- **video_20mph.mp4** video recording of the simulated car driven by trained model on track 1 at speed 20mph
- **video_reverse_direction.mp4** video recording of the simulated car driven by the trained model on track 1 in reverse direction (clockwise) at speed 15mph

All above mentioned project recourses are can be found here.

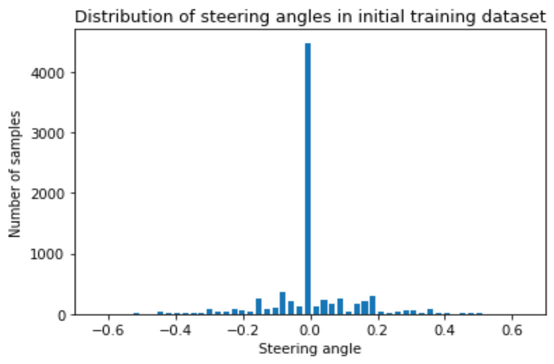
All visualizations used in this writeup document are implemented with means of matplotlib.pyplot library. Corresponding source code can be found among submitted project resource in training_data_visualization.ipynb file.

Training Data Collection

For this project Udacity provided the initial training data set that can be used only for initial model training since contains only the data samples of 'best case' scenario simulation: constant driving the center line on the simulated race track. This dataset, as well as other datasets additionally collected by me (see below) was generated using the driving simulator also provided by Udacity for this project. Each way point stored in a dataset contains images from 3 cameras located on center, left and right side of the simulated car. The description file driving_log.csv that is provided together with each such generated dataset contains following information about every recorded waypoint of the driven route:

Center Image	Left Image	Right Image	Steering Angle	Throttle	Break	Speed
IMG/center_2016_10_21_	IMG/left_2016_10_21_17_	IMG/right_2016_10_21_1	0	0.09803098	0	0
IMG/center_2016_10_21_	IMG/left_2016_10_21_17_	IMG/right_2016_10_21_1	0	0.09803098	0	0.06057268

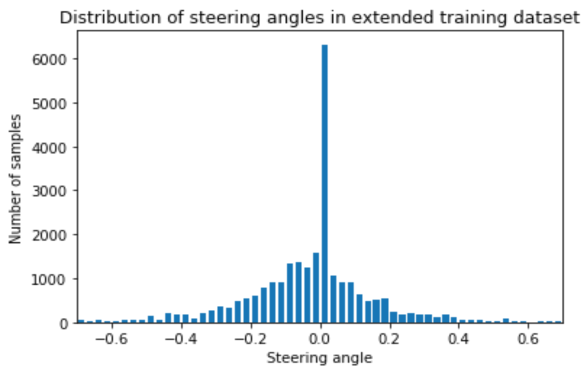
Following is the visualization of distribution of steering angles in provided dataset:



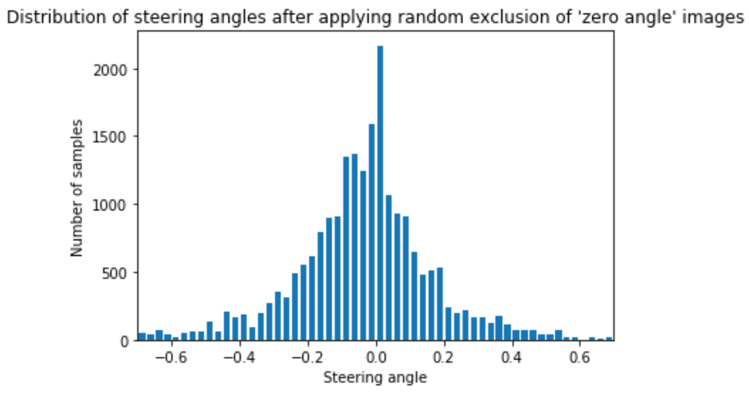
It clearly shows the complete domination of the images that correspond to zero steering angle driving, thus to decrease the possible bias of trained network weights caused by such non linear distribution of training samples, I decided to collect additional data (using a provided driving simulator in the training mode) to get more training samples for following driving scenarios:

- ideal driving lines for all of the track curves
- recoveries from different ‘driving out of the track’ situations around the track
- driving the track in reverse direction

After extending the provided dataset with collected samples the amount of non zero steering angle images in it has significantly increased (see the histogram below), though most of the added samples correspond to negative steering angle (turning to the left), since the default driving direction on the looped track 1 is counterclockwise

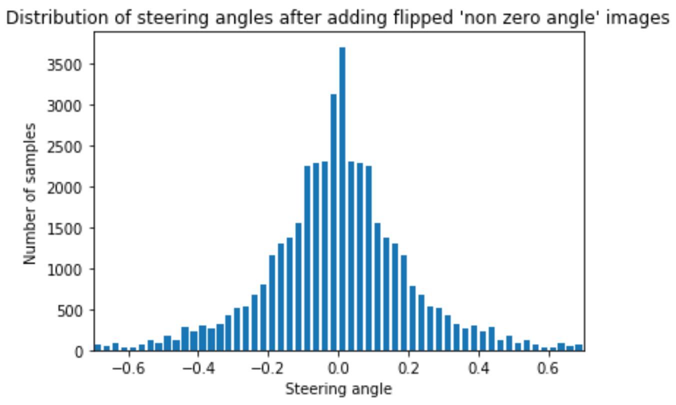


Also I decided to randomly exclude the zero steering angle images from the data set to decrease the ‘straight driving’ learning bias even more. For random exclusion I used the triple coin flip approach that has shown the most satisfying exclusion results:



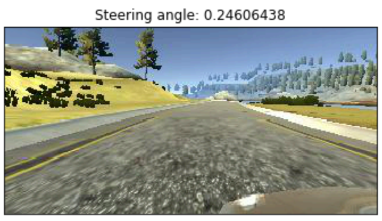
Training Data Augmentation

To compensate the bias of ‘mostly turning to the left‘ (domination of images that correspond to negative steering angles) and make the whole distribution of the steering angles in the dataset to look more like normal (Gaussian) distribution, I decided to use image flipping augmentation technic to the original images of dataset and extend it with resulting flipped images (lines 42 – 45 in model.py). This gave the following update in the steering angles distribution histogram:

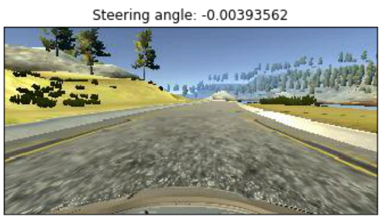


All above statistics of steering angles correspond to the images captured from center camera. But each recorded waypoint in the data set also contains the images captured from two additional cameras: located on the left and right sides of the car and rotated slightly to the center line of the car correspondingly. In order to increase the amount of training data even more I will include these images into the data set. To calculate the steering angles that would correspond to images from these additional cameras I will apply compensation approach to steering angle of ‘center camera image’ using following rule: add correction constant (0.25 has shown the best results) to ’center camera’ steering angle for image from left camera, and subtract steering angle correction constant from ’center camera’ steering angle for image from right camera (lines 59 – 63 in model.py). Below are displayed the images captured simultaneously from 3 different cameras that represent the same track waypoint.

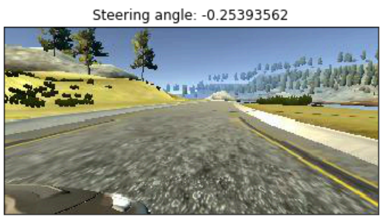
Left camera image



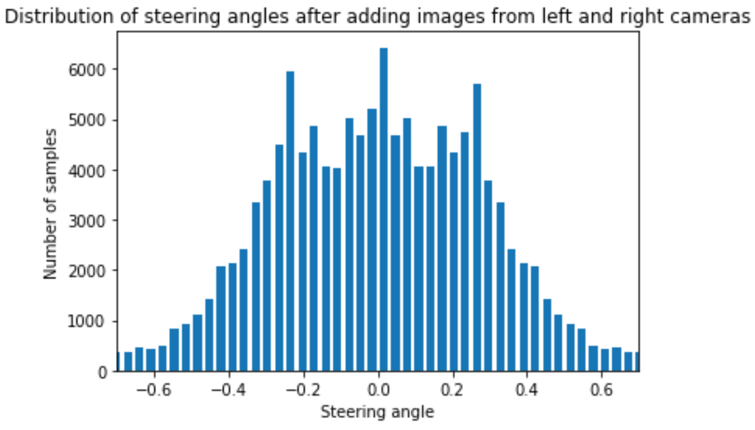
Center camera image



Right camera mage



After adding the left and right camera images to the dataset, the histogram of distribution of corresponding steering angles has following shape

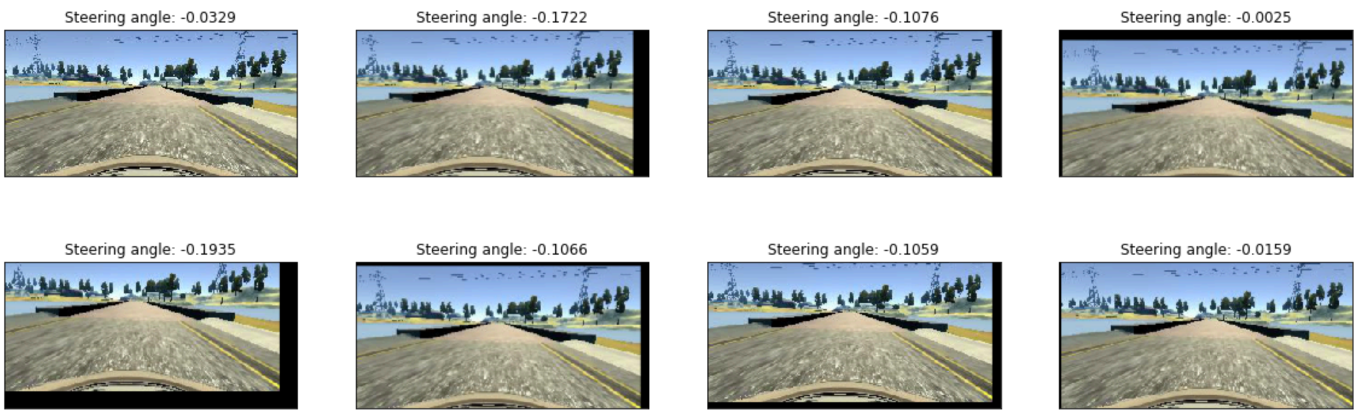


And the last augmentation technic that I will use for increasing the variety of my training dataset is the shifting of original images to random amount of pixels (using Affine transformation) within defined vertical and horizontal shifting ranges (after testing several different values for max shift ranges, 25 pixel max for horizontal shift (left or right) and 20 pixel max for vertical (up or down) have shown best results for me). Also together with pixel shifting I need to calculate new steering angle for such transformed image. For this I use following formula:

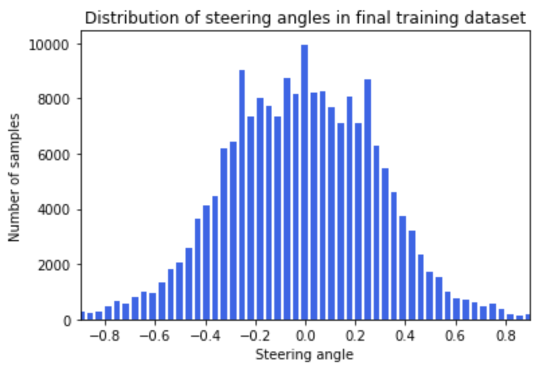
*new_steering_angle = initial_steering_angle + compensation_factor_of_one_pixel_shift * number_of_horizontally_shifted_pixels*

where *compensation_factor_of_one_pixel_shift = 0.4 / max_shift_range ,*

where 0.4 is the angle adjustment for maximal shift horizontally (right or left), also selected empirically (lines 15 – 24 in model.py). Here are several examples of the shifting the same image by random number of horizontal and vertical pixels:

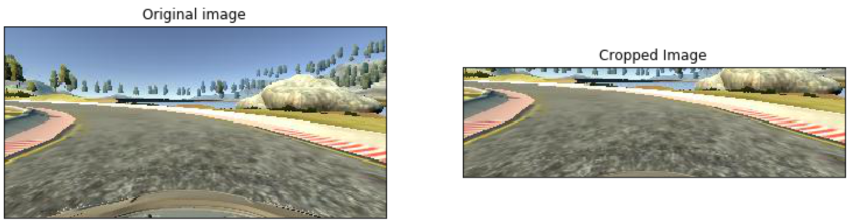


After adding shifted images to the training dataset (lines 38 – 40 in model.py), the final histogram of steering angles distribution has following shape:



Training data preprocessing

Original size (160 height x 320 width) of the images in training and validation datasets will be cropped (cropping will done 'inplace', as a first layer of the model, line 102 in model.py) before feeding into the network: 50 pixels from the top and 18 pixels from the bottom will be cut out to remove the content that does not bring any value for model training (like upper part of the landscape in the top of the image and most of the car hood in the bottom)This preprocessing step will allow to increase the speed of model learning and decrease the amount of used memory. Below is displayed the original and cropped image captured by center camera at random waypoint on the race track:



Additionally, in order to achieve near zero mean and equal variance in a range of [-0.5 : 0.5] across pixels of each image of training and validation dataset I will apply normalization of pixels for each channel using following formula (also done 'inplace', as second later of model, line 103 in model.py):

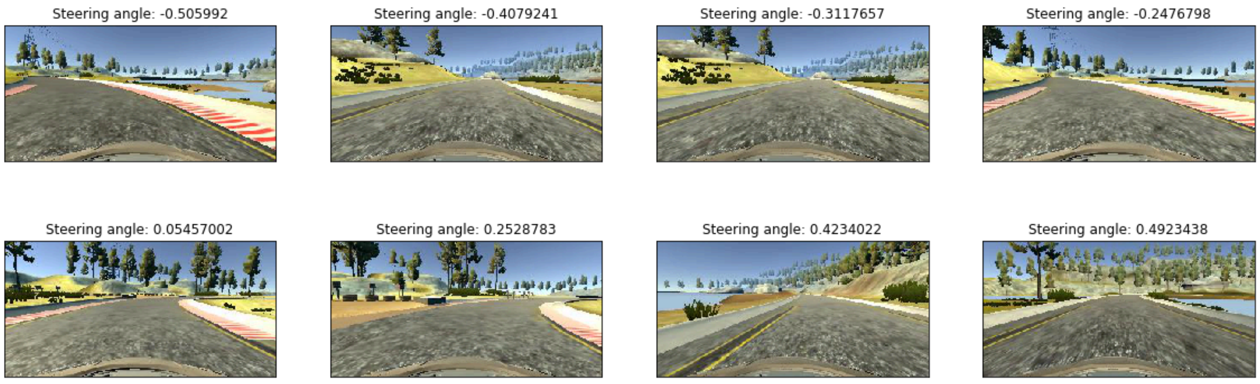
```
        normalizaed_pixel_value = pixel_value/255 – 0.5
```

This preprocessing step should speed up the overall training process.

I split the whole image and steering angle dataset into a training and validation set in 80/20 proportion. The training dataset will be used correspondingly for model training, and validation set will help to determine if the model is over or under fitting. Here are the statistics summary of final data sets:

- Number of images in training dataset: 156690
- Number of images in validation dataset: 39177
- The shape of images in both datasets: 160x320x3 (160x320 pixels widthxheight x 3 (RGB) channels)

Below images represent the random selection of samples from training dataset, that correspond to different steering angles from the range [-0.5 : 0.5]

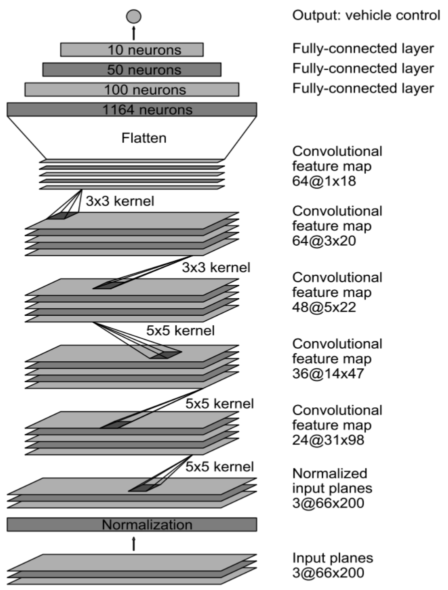

Steering angle: -0.505992
Steering angle: -0.4079241
Steering angle: -0.3117657
Steering angle: -0.2476798
Steering angle: 0.05457002
Steering angle: 0.2528783
Steering angle: 0.4234022
Steering angle: 0.4923438

Model Architecture

For selecting the most suitable model architecture for this project I decided to first try to use provided initial dataset for learning already existing solutions of CNN architectures:

- LeNet CNN architecture
- Self-Driving-Car control model from NVidia
- Steering angle prediction model from comma.ai.

LeNet and comma.ai models did not show good result for me: car was driving out of the track shortly after the start of simulation (LeNet right away, comma.ai after a couple of hundred meters). On other hand the model from NVidia displayed on the picture below, which was trained on initially provided training dataset, has shown pretty good ability of controlling the car along the whole simulated track (track 1).



But because of huge amount of training parameters (network has about 27 million connections and 250 thousand parameters) and big size of the final state of training data set, the training even a single epoch was taking significant amount of time even on Amazons instance of g2.2xlarge GPU. Additionally to that persisted trained NVidia model was taking about 150MB of disc space which was coursing me a trouble of storing the trained model in a GitHub repository as part of required project resources. Thus I decided to experiment with decreasing the initial NVidia's model to the size that would be small enough for fast learning on my dataset and would still fulfill my requirements for safe and smooth driving of the car through the race track.
After various experiments with removing one to several convolutional and/or fully connected layers from original NVidia model, as well as modifying original hyperparameters for sizes of layers, I ended up on decision to remove only the last convolutional layer (64 kernels of 3x3) and keep the original hyperparameters unchanged, since other tried alternatives of the model architecture did not show very good results on the track

The final model architecture represented here is implemented in python using Keras library (lines 101-114 in model.py) and consists of the following layers:

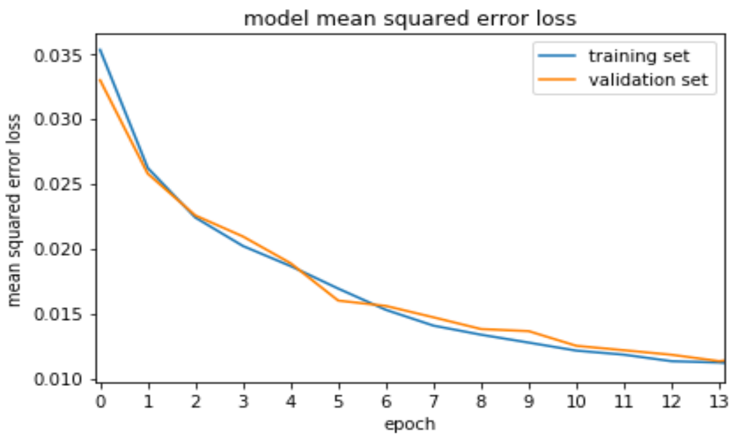
Layer	Description	Input	Output
Crop	cropping of original images to size 92x320	160x320x3	92x320x3
Lambda	normalization of pixel values to range [-0.5 : 0.5]	92x320x3	92x320x3
Convolution 5x5	depth:24, stride: 2x2, padding: valid, activation: ReLU	92x320x3	44x158x24
Convolution 5x5	depth:36, stride: 2x2, padding: valid, activation: ReLU	*44x158x* 24	20x77x36
Convolution 5x5	depth:48, stride: 2x2, padding: valid, activation: ReLU, dropout: 30%	*20x77x*36	*8x37x*48
Convolution 3x3	depth:64, stride: 1x1, padding: valid, activation: ReLU, dropout: 30%	*8x37x*48	*6x35x*64
Flatten	transforms 3D matrix into 1D array	*6x35x*64	13440
Fully connected	size 100, interconnects all inputs to all outputs	13440	100
Fully Connected	size 50, interconnects all inputs to all outputs	100	50
Fully Connected	size 10, interconnects all inputs to all outputs	50	10
Fully Connected	size 1, interconnects all inputs to single output	10	1

Trained model of my truncated version of NVidia network architecture is consuming only 17.4 MB in persisted state on the hard disk.

Model training strategy

After removing above mentioned layer form original NVidia model, I found that my 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 overfitting. To combat the overfitting, I decided to add dropout for two last convolution layers with dropout percentages 30% and 30% accordingly (selected empirically after trying several values in a range [20%-50%]).
To limited the model training process to 14 epochs, since additional epochs of training did not bring any more decrease for validation and training loss.
The batch size for training of each epoch was 288 and defined in following way: 32 track waypoints x multiplication factor of one waypoint data augmentation, which is 9: (views from 3 cameras x (original image + number of augmented images (flipped and shifted)).
For model optimization I used Adam optimizer thus manual tuning of learning rate wasn't necessary.

Following is the visualization of how the training and validation Mean Squared Error (MSE) loss of the model was decreasing over the epochs of training.



Testing of the model and analysis of results

As can be seen from above visualization, adding dropout allowed to remove the model overfitting effect and get validation and training loss values almost identical after 14 epoch of training.

All model evaluations have been done in provided simulator application in autonomous mode with following UI settings:

- Screen resolution: 800x600
- Graphics quality: Fast

The final version of trained model was showing such a great control of the car on race track at default configured speed (9 mph), that I decided to increase the speed to 15 mph by modifying corresponding value in the drive.py script. Even on incerased speed the car was constantly trying to keep the center lane on the straight segments and smoothly passing the curves. Also difficult parts of the track start of the bridge with bumpy surface and shadowed segments did not cause any big troubles for the model. Video recording of this test can be found on video.mp4 file.
The simulator provides the option to intervene with manual control even in autonomous mode. I have used this ability to cause several hazardous ‘running out of the track‘ situation and test the ability of the model to recover from them. As I was expected, since the training data contained big portion of ‘recovery’ samples recorded on various segments on the track, the model successfully recovered from all ‘gotcha’ cases (to certain level of severity of course)
Another test which I have applied was testing the model for controlling the car while driving in a opposite to default direction on the track (clockwise). To make it possible, I again used the option of manual steering of the car to turn it into opposite direction before running the model. Since the training data also contained samples recorded while driving in the opposite direction on the track, this task was also not a big deal for my model (results of this experiment can be seen in submitted video file video_reverse_direction.mp4)
Optionally I decided to increase the speed even more from 15mph to 20mph in order to test the ability of the model to control the car in more challenging conditions, since it would require even faster reaction from the model on more rapidly changing road environment. As can be seen in submitted file video_20mph.mp4, the increase of the speed has caused the some oscillations of the car in various segments of the track, but the car still successfully finished the full lap.
Increasing the speed to 25mph though started to trigger the oscillations of such a high levels, that they were consistently causing the car running out of the track in a very short distance after the start. This opens the possibility for further

improvements of the model. Most obvious first step in this direction would be collecting of the training data while driving the car in training mode with corresponding high speed (especially important would be the recording of big amount of ‘recovery’ samples for potential ‘running out of the track‘ situations at such high speed).

All above testing results have proven the big importance of the quality and variety of used training data samples for getting the high accuracy of predictions from trained model in different testing scenarios and ability of the model to adapt to unknown input data (generalize)