
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

Rubric Points

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

###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.ipynb containing the script to create and train the model
- model.py (same as above, just to meet the project requirement).
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolutional 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
```

####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. Note- I developed it Jupyter notebook and later copied model.ipnb to model.py to meet the project requirement.

###Model Architecture and Training Strategy

####1. NVIDIA model architecture has been employed

I followed the pattern taught in the class so I started with GoogleLeNet (model.py line no 209-217) and ended with using Nvidia (line no 220-230).

My model consists of a convolution neural network having 3 convolution layers with 5x5 filters and another 2 convolution layers with 3x3 filters.

Layers Details

The input shape is 160*320*3 and the

- Input shape 160*320*3
- Keras Lambda normalized layer mean centered.
- Crop layer to remove the unwanted portion (50%) from the top (sky etc) and (20%) from the bottom (car hood)
- The 1st convolution layer with kernel size 5*5, stride 2*2 and depth 24
- The 2nd convolution layer with kernel size 5*5, stride 2*2 and depth 36
- The 3rd convolution layer with kernel size 5*5, stride 2*2 and depth 48
- The 4th convolution layer with kernel size 3*3, stride 1 and depth 64
- The 5th convolution layer with with kernel size 3*3, stride 1 and depth 64
- Flatten layer
- The 1st fully connected layer of output size 100
- The 2nd fully connected layer of output size 50
- The 3rd fully connected layer of output size 10
- Final fully connected layer of output size 1

####1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 3x3 filter sizes and depths between 32 and 128 (model.py lines 18-24)

The model includes RELU layers to introduce nonlinearity (code line 20), and the data is normalized in the model using a Keras lambda layer (code line 18).

####2. Attempts to reduce overfitting in the model

Dataset have been split into training and validation set with the validation set to 20%.

Data are shuffled so that it should not be overfitting. The model was tested by running it

through the simulator and ensuring that the vehicle could stay on the track.

####3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 232).

####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

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

###Model Architecture and Training Strategy

####1. Solution Design Approach

I followed the pattern taught in the class i.e started with simple and kept improving it with every iteration. I generated data running the simulator provided in the "training" mode. It was only after 4-5 iteration that I was able to drive the car manually through computer arrow keys. Finally I ended up recording the data for about 2 laps.

The overall strategy for deriving a model architecture was to ...

My first step was to use a convolutional neural network model similar to the ... I thought this model might be appropriate because ...

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 overfitting.

To combat the overfitting, I modified the model so that ...

Then I ...

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... to improve the driving behavior in these cases, I ....

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

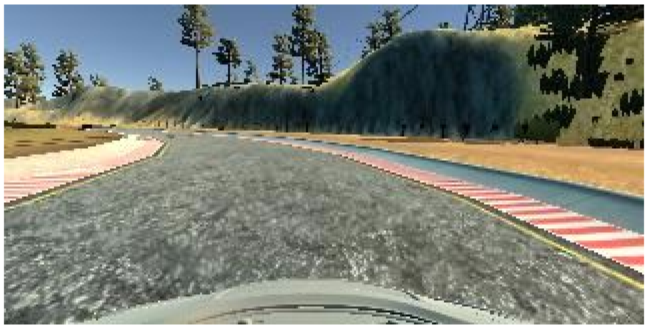
####2. Final Model Architecture

The final model architecture (model.py lines 18-24) consisted of a convolution neural network with the following layers and layer sizes ...

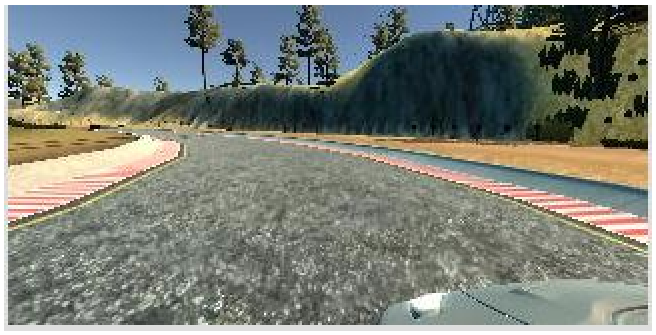
####3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. It requires 10-15 times Here is an example image of center, left and right lane driving at the same time.

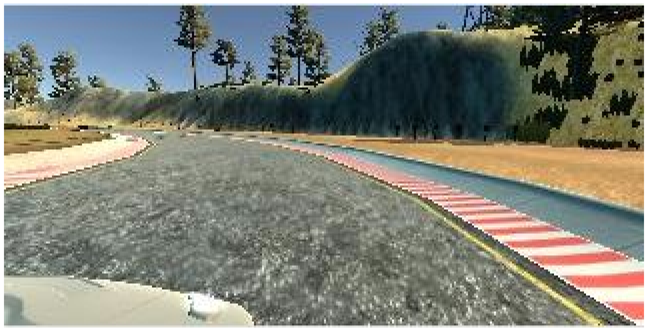
Image captured from center camera



Example Image captured from the left camera



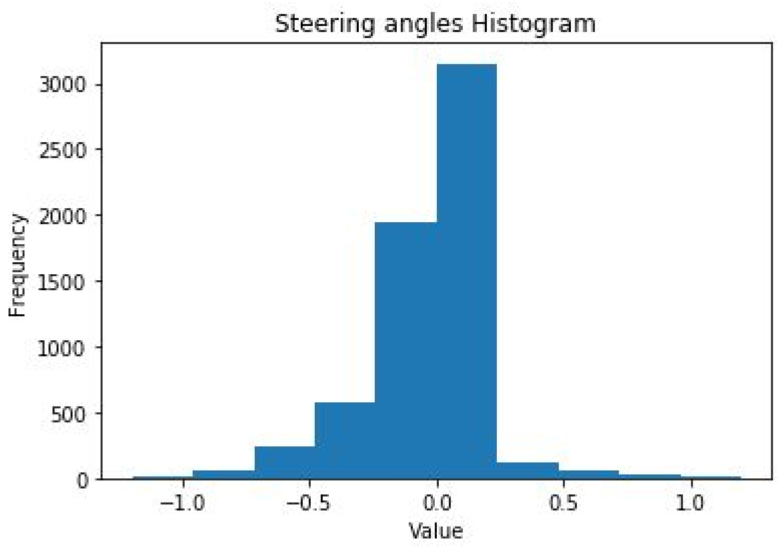
Example Image captured from the right camera



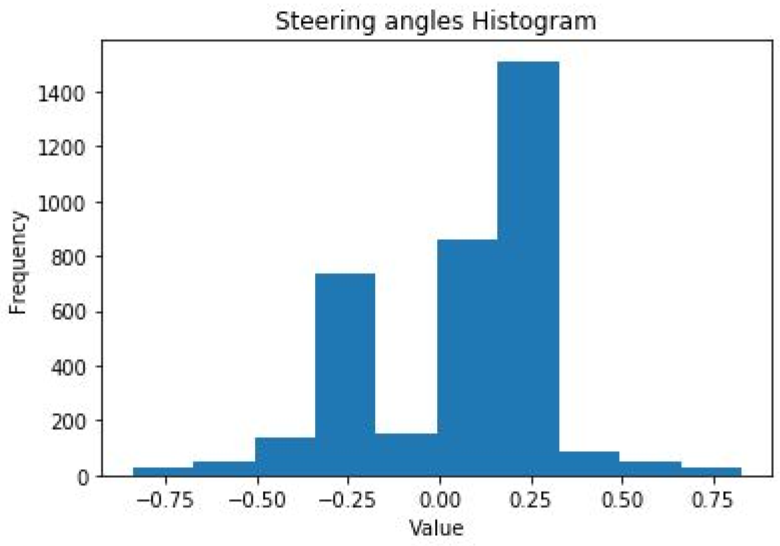
Lack of time I could not repeat this process on track two. Accumulating data from track 2 could have further improved the results.

To augment the data sat, I also flipped the images (line no 173-175) and angles thinking that this would increase the distribution of the data.

After the collection process, I had 6198 number of data points.



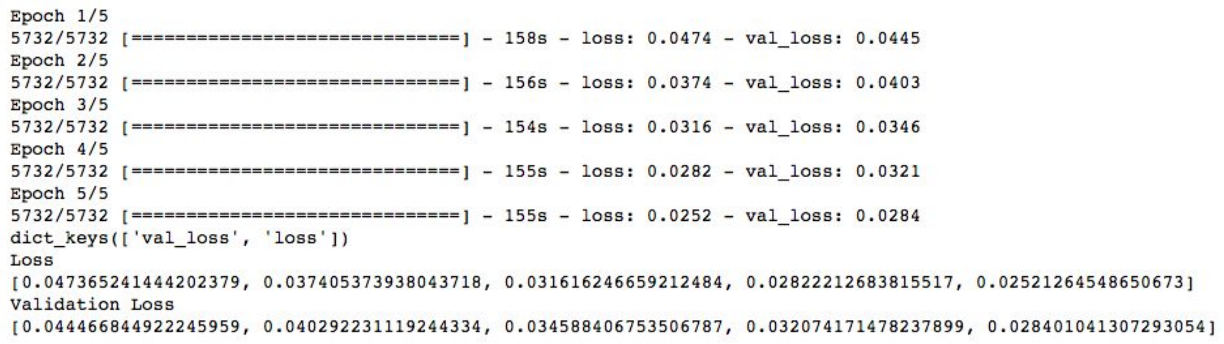
I filtered out 2571 data points to offset the bias towards 0 steerings.



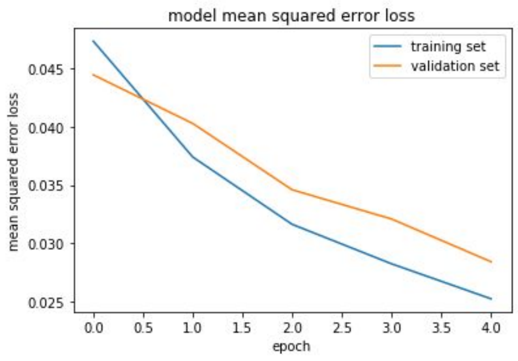
```
Filtered data
```

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

I executed this several times tuning the parameters. Here is the plot for the mean squared error loss for training and validation data set.



```
Epoch 1/5
5732/5732 [==============================] - 158s - loss: 0.0474 - val_loss: 0.0445
Epoch 2/5
5732/5732 [==============================] - 156s - loss: 0.0374 - val_loss: 0.0403
Epoch 3/5
5732/5732 [==============================] - 154s - loss: 0.0316 - val_loss: 0.0346
Epoch 4/5
5732/5732 [==============================] - 155s - loss: 0.0282 - val_loss: 0.0321
Epoch 5/5
5732/5732 [==============================] - 155s - loss: 0.0252 - val_loss: 0.0284
dict_keys(['val_loss', 'loss'])
Loss
[0.047365241444202379, 0.037405373938043718, 0.031616246659212484, 0.02822212683815517, 0.025212645486506731]
Validation Loss
[0.044466844922245959, 0.040292231119244334, 0.034588406753506787, 0.032074171478237899, 0.028401041307293054]
```



Link for my video is
https://github.com/2sbsbsb/CarND-Behavioral-Cloning-P3/blob/master/run2.mp4
Or on youtube
https://www.youtube.com/watch?v=q5KwdbeYPZs