
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. The line no in this document refers to model.ipynb
- model.py (same as above, just to meet the project requirement).
- video-with-opencv-3.ipynb.
- 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 Google LeNet (line no 47-54-commented out) and ended with using Nvidia (line no 57-69).

```
57 # NVDIA
58 reg = 1e-3
59 model.add(Convolution2D(24,5,5,subsample=(2,2),activation="relu"))
60 model.add(Convolution2D(36,5,5,subsample=(2,2),activation="relu"))
61 model.add(Convolution2D(48,5,5,subsample=(2,2),activation="relu"))
62 model.add(Convolution2D(64,3,3,activation="relu"))
63 model.add(Convolution2D(64,3,3,activation="relu"))
64 model.add(Flatten())
65 model.add(Dropout(0.5))
66 model.add(Dense(100,activation="relu",W_regularizer=l2(reg)))
67 model.add(Dense(50,activation="relu",W_regularizer=l2(reg)))
68 model.add(Dense(10,activation="relu",W_regularizer=l2(reg)))
69 model.add(Dense(1))
```

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

####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. To reduce the overfitting L2 regularization has been used (line no 65-68).

```
65 model.add(Dropout(0.5))
66 model.add(Dense(100,activation="relu",W_regularizer=l2(reg)))
67 model.add(Dense(50,activation="relu",W_regularizer=l2(reg)))
68 model.add(Dense(10,activation="relu",W_regularizer=l2(reg)))
```

####3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.ipynb line 71).

```
71 model.compile(loss="mse", optimizer="adam")
```

####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 first used Google LeeNte but ended up commenting it and later used NVIDIA architecture.

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. I also tried the 2nd track but could not able to keep the car on the track. I even tried using the data provided in the class.

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

####2. Final Model Architecture

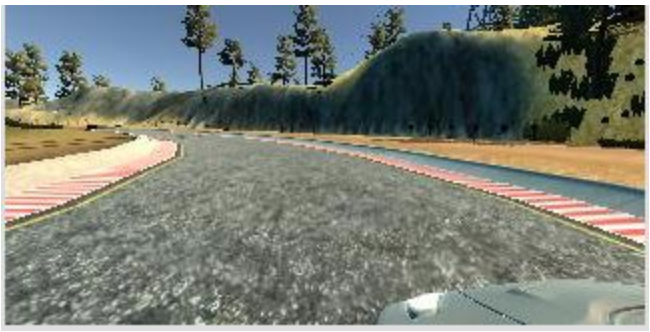
Please refer to this section “NVIDIA model architecture has been employed” above.

####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

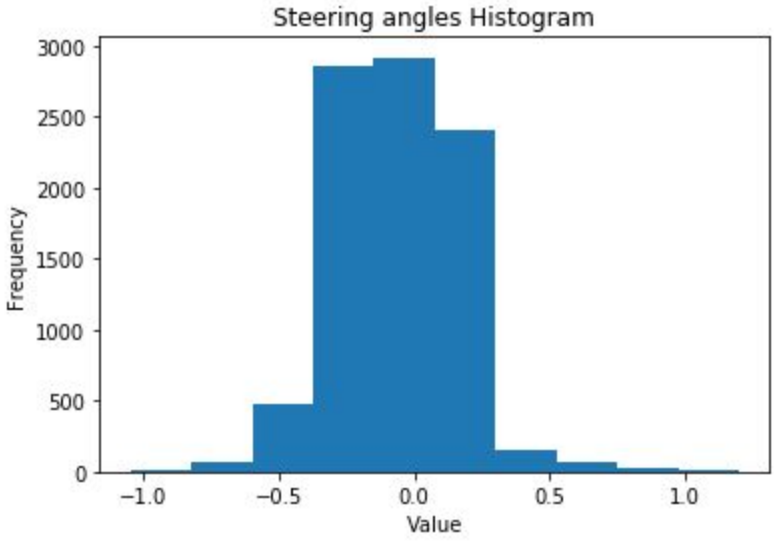


Example Image captured from the right camera

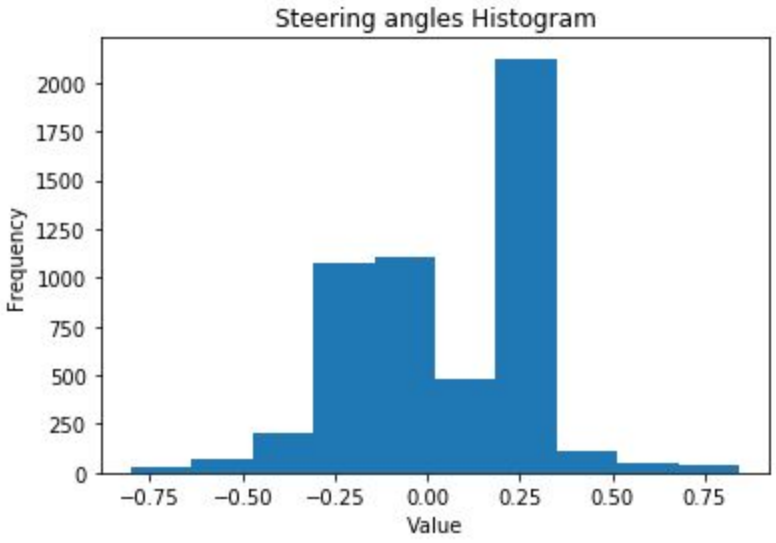
Lack of time I could not repeat this process on track two. I must say I tried but but failed miserably while training the data, the car would almost everytime came out of the track. Given the time I had spent on track 1 (weeks). I gave up the track 2. However accumulating data from track 2 could have further improved the results.

To augment the data sat, I also flipped the images (line no 27) and angles thinking that this would increase the distribution of the data.

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



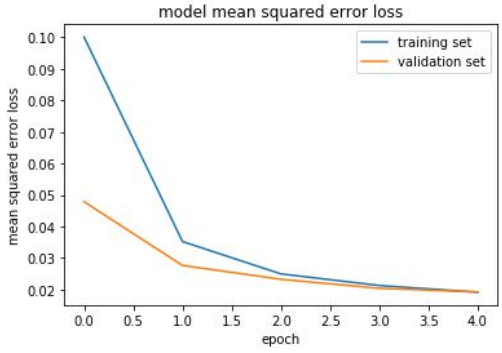
I filtered out 3742 data points to offset the bias towards 0 steerings.



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
8454/8454 [==============================] - 241s - loss: 0.1001 - val_loss: 0.0479
Epoch 2/5
8454/8454 [==============================] - 238s - loss: 0.0352 - val_loss: 0.0277
Epoch 3/5
8454/8454 [==============================] - 238s - loss: 0.0250 - val_loss: 0.0233
Epoch 4/5
8454/8454 [==============================] - 238s - loss: 0.0213 - val_loss: 0.0204
Epoch 5/5
8454/8454 [==============================] - 238s - loss: 0.0192 - val_loss: 0.0192
dict_keys(['loss', 'val_loss'])
Loss
[0.100078692942195, 0.035228453687609636, 0.024965016141371692, 0.021285134110302754, 0.019152337331822755]
Validation Loss
[0.047859052102748469, 0.027650641769086177, 0.023258116663686487, 0.020392979048509698, 0.019225780227439 18]
```



I received few comments during code-review and one of the suggestion was to remove overfitting and after using L2 regularization and dropout i can see that overfitting issue has been addressed.

Link for my video is https://youtu.be/eLJPl5CpQEI