
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

[//]: # (Image References)

[image1]: ./examples/placeholder.png "Model Visualization"
[image2]: ./examples/placeholder.png "Grayscaling"
[image3]: ./examples/placeholder_small.png "Recovery Image"
[image4]: ./examples/placeholder_small.png "Recovery Image"
[image5]: ./examples/placeholder_small.png "Recovery Image"
[image6]: ./examples/placeholder_small.png "Normal Image"
[image7]: ./examples/placeholder_small.png "Flipped Image"

Rubric Points
###Here I will consider the [rubric points](https://review.udacity.com/#!/rubrics/432/view) 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.py containing the script to create and train the model
* drive.py for driving the car in autonomous mode
* model.h5 containing a trained convolution 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.json
```

####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.

###Model Architecture and Training Strategy

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

My model consists of a convolution neural network with 5x5 filter sizes and depths between 16 and 64 (model.py lines 50-90)

The model includes multiple RELU layers to introduce nonlinearity

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

The model contains dropout layers in order to reduce overfitting (model.py lines 71 and 85).

The model includes a max pooling layer with the pool size 2x2 (model.py line 69) – also for the purposes of reducing the number of parameters

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 103). 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 96). However, multiple other hyperparameters were tuned, as described later in this document.

####4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road – a task I found to be surprisingly challenging. I used the following driving styles in the final data set:

- center lane driving
- center lane driving in opposite direction
- recovering from left and right sides of the road (but within the lane markers)
- emergency recovering from the curb areas left and right with very high steering angle (beyond the lane markers)

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

###Model Architecture and Training Strategy

####1. Solution Design Approach

The overall strategy for deriving a model architecture was to start with a simpler, more ‘shallow’ network and only add layers as absolutely necessary, while simultaneously employing dropout and max pooling to reduce overfitting and number of parameters.

My first step was to use a CNN with just two convolutional layers and a single dropout layer. The filters used were never particularly deep – max depth was 8.

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 very high accuracy (compared to my later models) – but later found this was due to my method of data collection. As my first data set was collected using keyboard input only, majority of angles were either 0 or -/+ 25.0. This made the prediction of 'floating point' steering angle disproportionately accurate (~about 70%). I describe how I combated this later.

In my later versions, I added a few more convolutional layers as it seemed the model does not have enough parameters to really generalize the behavior. Soon, however, I found that the model was growing disproportionately large – at about 1.5 million trainable parameters it felt like an overkill for my available training data. To combat this, I empirically ran multiple versions, tweaking the hyper parameters, esp. the dropout values and filter depths + patch sizes, until I found an architecture that had loss on validation set of just under 1% (and accuracy of about 7%)

The final step was to run the simulator to see how well the car was driving around track one. There was one spot where the vehicle drove off the track – the sharp right turn. Collecting just a few additional seconds of recovery and emergency recovery data around the track area solved the problem.

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:

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 20, 80, 16)	416	convolution2d_input_1[0][0]
activation_1 (Activation)	(None, 20, 80, 16)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 8, 38, 32)	12832	activation_1[0][0]
activation_2 (Activation)	(None, 8, 38, 32)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 2, 17, 64)	51264	activation_2[0][0]
activation_3 (Activation)	(None, 2, 17, 64)	0	convolution2d_3[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 1, 8, 64)	0	activation_3[0][0]
dropout_1 (Dropout)	(None, 1, 8, 64)	0	maxpooling2d_1[0][0]
flatten_1 (Flatten)	(None, 512)	0	dropout_1[0][0]
activation_4 (Activation)	(None, 512)	0	flatten_1[0][0]
dense_1 (Dense)	(None, 512)	262656	activation_4[0][0]
activation_5 (Activation)	(None, 512)	0	dense_1[0][0]
dense_2 (Dense)	(None, 16)	8208	activation_5[0][0]
activation_6 (Activation)	(None, 16)	0	dense_2[0][0]
dropout_2 (Dropout)	(None, 16)	0	activation_6[0][0]
activation_7 (Activation)	(None, 16)	0	dropout_2[0][0]
dense_3 (Dense)	(None, 1)	17	activation_7[0][0]

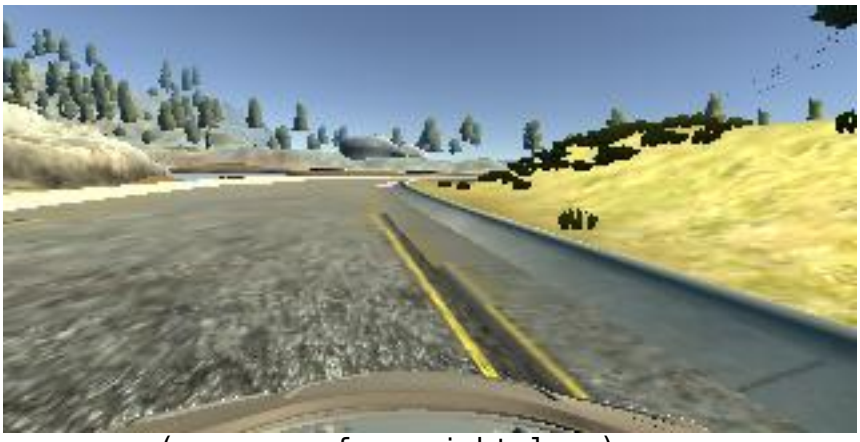
Total params: 335,393
Trainable params: 335,393
Non-trainable params: 0

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

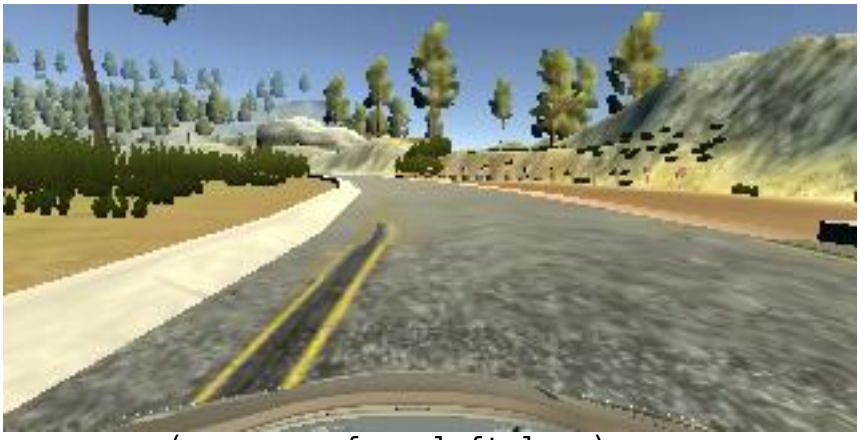
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded one lap of center driving, but in the opposite direction. I did this to reduce left turn bias, while generating unique training data – therefore being able to skip augmentation via flipping the images etc.

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 how to get back to the center if it drifts toward one of the lane markers. These images show what a recovery looks like from right and left lane respectively.



(recovery from right lane)



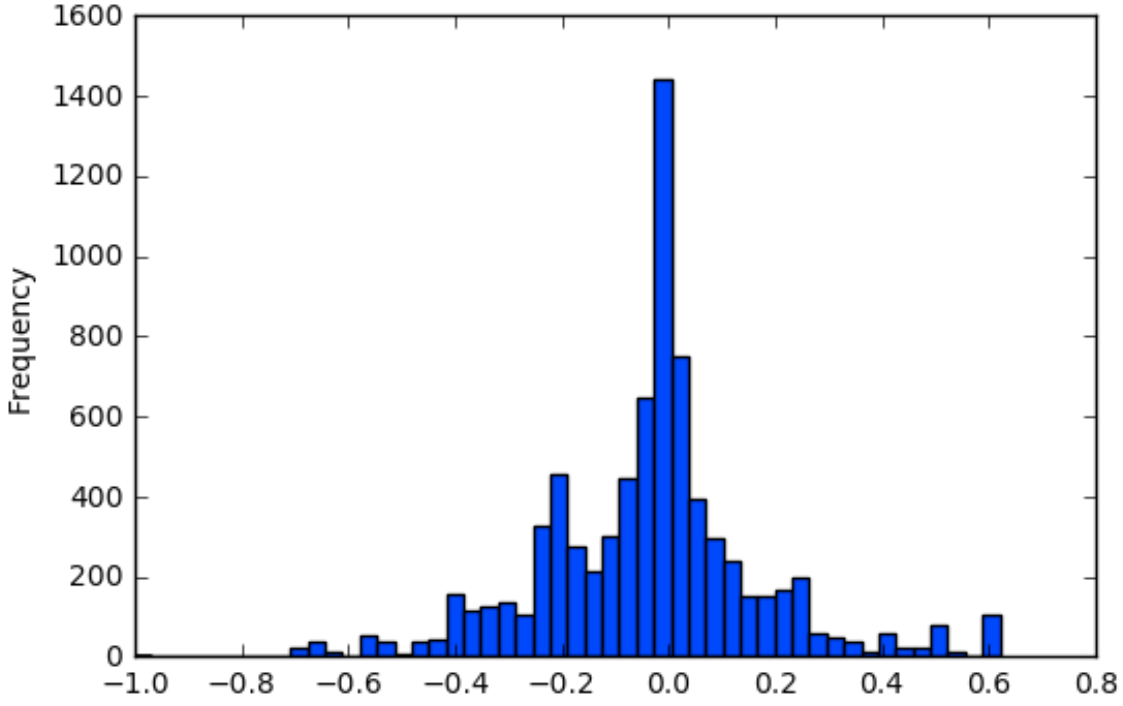
(recovery from left lane)

Finally, I recorded several seconds of ‘emergency recovery’, i.e. action that car needed to take if one of the wheels went beyond the lane marker and touched the curb. In this case I wanted the car to learn a sharp turn with high steering angle – but only briefly to prevent wobble / oversteer in the opposite direction. Here I aimed (and succeeded) at making the driving even more robust in non-standard situations.

(emergency recovery from curb)

After the collection process, I had 7816 number of data points. I further augmented this by using images from all three cameras for a total of 23448 images.One major shortcoming I have not addressed is changing the steering angle based on which camera was used – adding or subtracting 2-3 degrees to the steering angle depending on the camera would have made the model more robust.

This is the distribution of the data I was working with – there is a left skew, but significantly lower than if I did not drive the car in the opposite direction.



I also cropped & resized the images to size 20 by 80 and normalized them (/255 – 0.5). This made the model not only faster, but more accurate by removing the parts like trees and hood which were not aiding in the navigation and were potentially confusing the network.

I finally randomly shuffled the data set and put 10% of the data into testing dataset, and 20% of the remaining data into 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 about 12 (I ended up using 15 which was potentially too many), which was evidenced delta of loss approaching zero. I used an adam optimizer so that manually training the learning rate wasn't necessary.