ameter / **CarND-BehavioralCloning-P3**

CarND Project 3

Edit

Manage topics

36 commits	**1** branch	**0** releases	**1** contributor

Branch: master ▾ | New pull request | Create new file | Upload files | Find File | Clone or download ▾

ameter Update writeup_report.md — Latest commit `6acec00` on Dec 4, 2017

writeup_images	added mse loss charts	a year ago
README.md	renamed writeup to reflect that it is markdown	a year ago
drive.py	Modified drive.py to convert images from RGB (PIL format) to BGR (Ope…	a year ago
model.h5	Trained with 50 epochs on additional track-2 data.	a year ago
model.py	Trained with 50 epochs on additional track-2 data.	a year ago
train.log	trained	a year ago
video.mp4	recorded final run video	a year ago
video.py	initial commit	a year ago
writeup_report.md	Update writeup_report.md	a year ago

README.md

Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolutional 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.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.md 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 convolutional 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 convolutional neural network with kernel sizes between 3x3 and 5x5 and depths between 24 and 64 (model.py lines 79-96)

The model includes RELU activations to introduce nonlinearity (code lines 79-87), and the data is normalized in the model using a Keras lambda layer (code line 77).

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to determine if the model was overfitting (code lines 66-70). Dropout was added after every layer to further reduce overfitting (code lines 80-95). 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 98).

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, driving the course counter-clockwise, and flipping images during preprocessing.

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 leverage a published architecture and then adapt it as necessary.

My first step was to use a convolutional neural network model based on work published by nvidia: https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/

I thought this model might be appropriate because nvidia has used to to successfully drive actual cars.

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 relatively high mean squared error on both the training and validation set. This implied that the model was underfitting.

To combat the underfitting, I modified added a preprocessing step to normalize the data for zero mean and equal variance (code line 77).

Then I croped the tops and bottoms off the images to mostly show the road instead of the front of the car and surrounding scenery (code line 75).

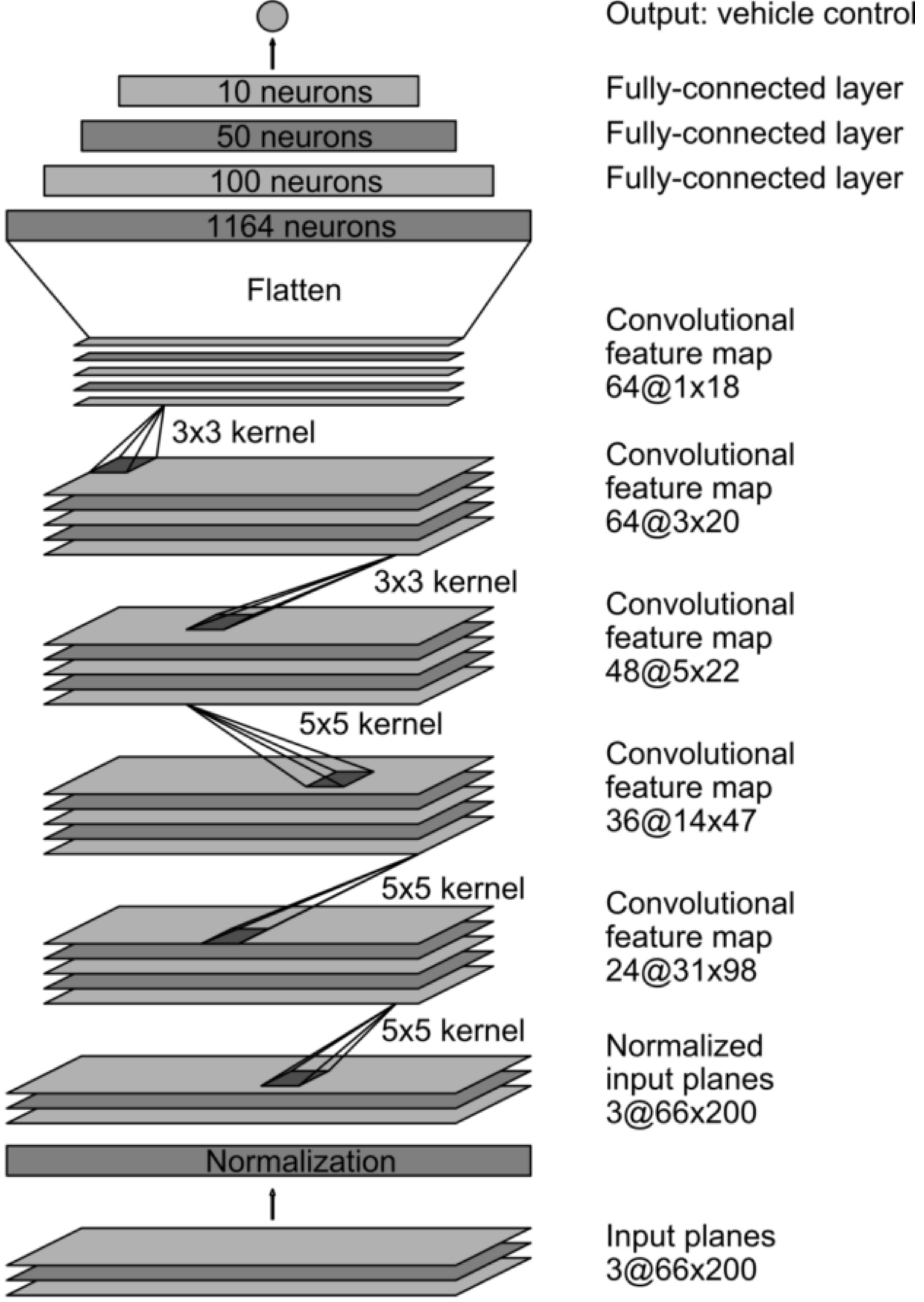
After addressing the underfitting issue, the model had a low mean squared error on the training set, but a relatively high mean squared error on the validation set. This implied that the model was overfitting. To address this, I added dropout after every layer of the model. I experimented with a number of different dropout rate and achieved good performance with a rate of 0.25.

The final step was to run the simulator to see how well the car was driving around track one. There vehicle tended to pull to the left and was unable to recover when it got off track. To improve the driving behavior in these cases, I recorded additional center lane driving, added images from the left and right camera with a steering correction of 0.15 (code line 38), generated revcovery images by drving near the edge of the track and then recording the recovery, and generated additional data that would not favor left turns by creating flipped images and steering angles (code lines 50, 52).

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

2. Final Model Architecture

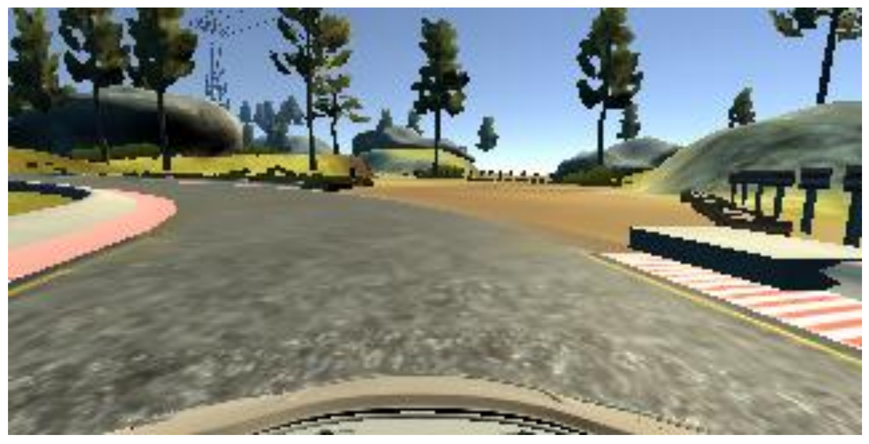
The final model architecture (model.py lines 75-96) consisted of a convolutional neural network with the following layers and layer sizes. Note, this visualization of the architecture was provided by nvidia:



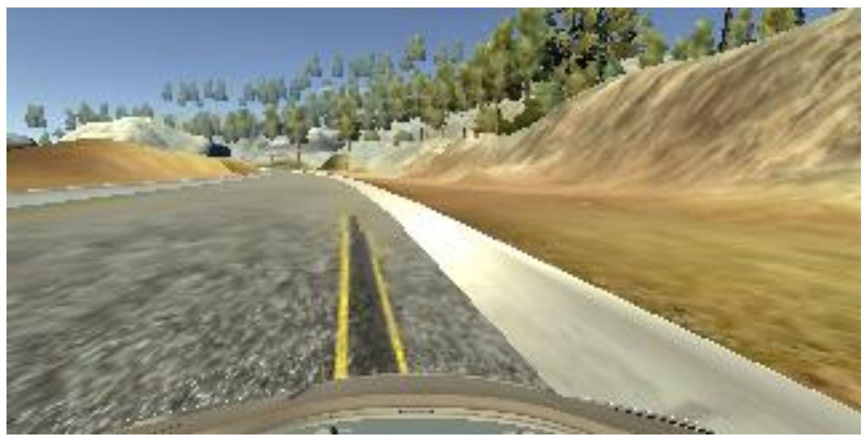
I added a dropout rate of .25 after every layer.

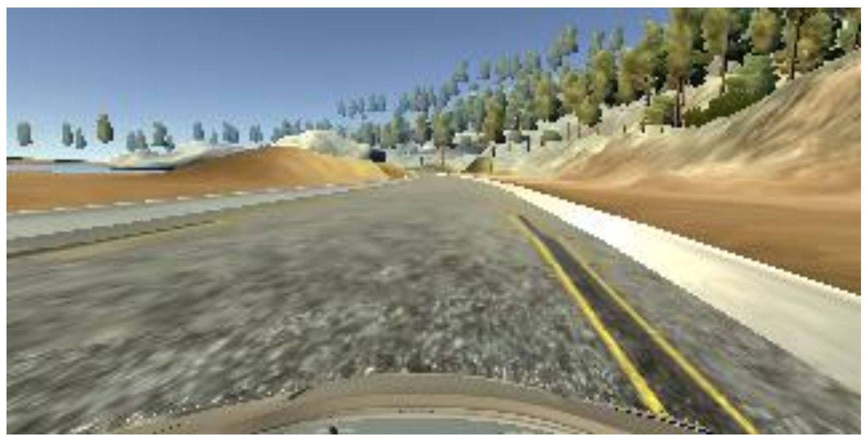
3. Creation of the Training Set & Training Process

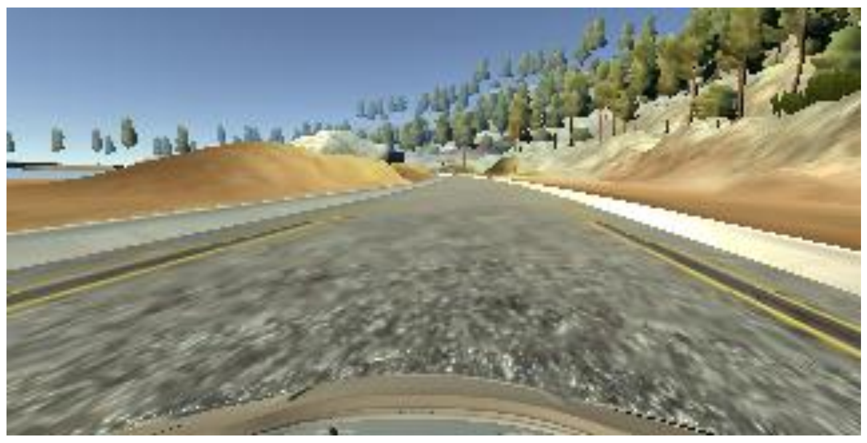
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:



I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover when it got near the edge of the track. These images show what a recovery looks like starting from right side of the track :







Then I repeated this process on track two in order to get more data points.

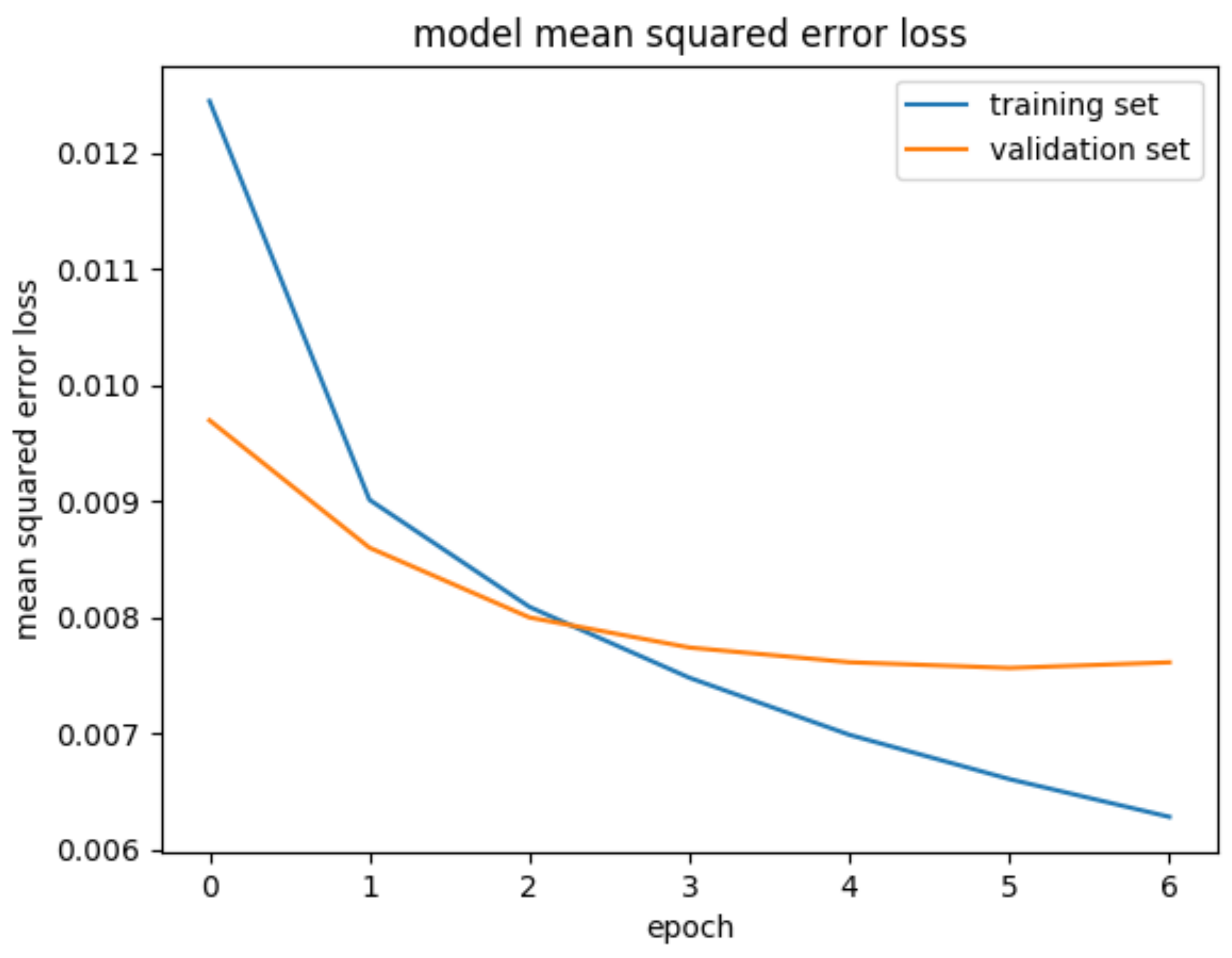
To augment the data set, I also flipped images and angles thinking that this would reduce the model's tendency to steer to the left.

After the collection process, I had 82,450 data points. I then preprocessed this data by applying the following function to every pixel: (pixel / 255.0) - 0.5

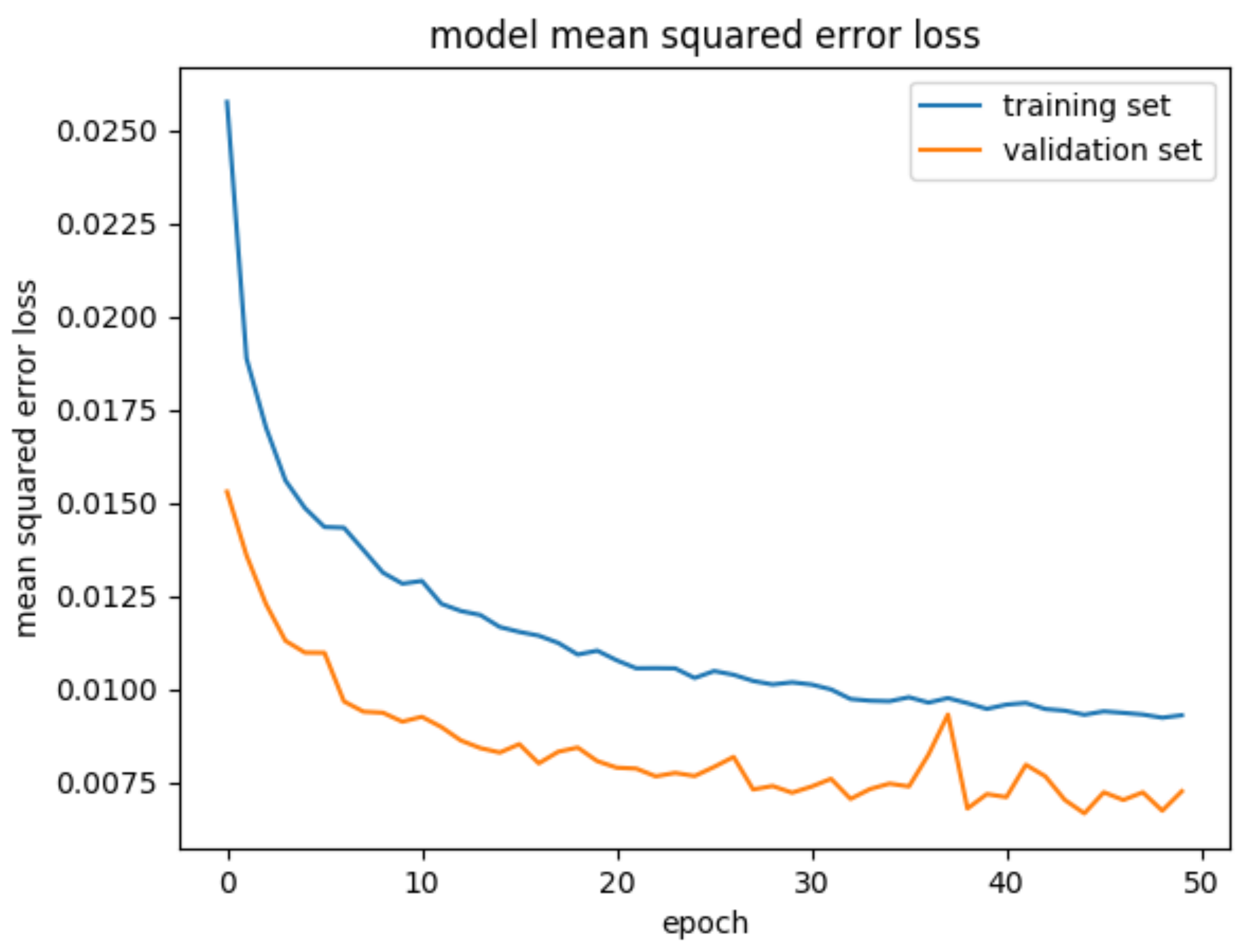
I finally randomly shuffled the data set and put 20% of the data into a validation set.

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 before adding dropout was between 4 and 7 (3 and 6 in the zero-based numbering seen below) as evidenced by the fact that the mean squared error (mse) loss on the validation set stopped decreasing beyond that. The ideal number of epochs after adding dropout was approximately 45. The following charts show mse loss over epochs.

Without Dropout



With Dropout



I used an adam optimizer so that manually training the learning rate wasn't necessary.