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 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
- Test that the model also successfully drives around track two 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 or writeup_report.pdf----summarizing the results
- video_track1.mp4----A video recording of my vehicle driving autonomously at least one lap around the track one.
- video_track2.mp4----A video recording of my vehicle driving autonomously at least one lap around the track two.

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.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

I referred to Nvidia's convolutional neural network for self-driving cars from the paper--End to End Learning for Self-Driving Cars and Comma.ai's steering angle prediction model.
My model consists of a convolution neural network based on Nvidia's model architecture(model.py lines 124-145) but slightly different: I add a Cropping2D layer to crop the input image and I add some dropout layers to generalize the model better.

The model includes RELU layers to introduce nonlinearity (code line 130-134, 137, 139, 141), and the data is normalized and mean centered in the model using a Keras lambda layer (code line 129).

For details about the final chosen appropriate model, see the next section(Final Model Architecture).

2. Attempts to reduce overfitting in the model

The model contains 4 dropout layers in order to reduce overfitting (model.py lines 136, 138, 140, 142).

The model was trained and validated on different data sets(both driving data for track 1 and track 2) to ensure that the model was not overfitting (code line 161-169).

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 tuned automatically (model.py line 173).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, driving in a counter-clockwise direction, and using both track 1 and track 2 driving behavior data.

For data augmentation, I also used multiple cameras' images and flipped the images and steering measurements.

I filtered about 90% driving data samples with 0 degree steering angle, because they were overrepresented and made the datasets imbalanced.

For details about how I created the training data, see the next section(Creation of the Training Set & Training Process).

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to use CNN (short for convolutional neural network) and followd by a few fully connected layers. The Nvidia's paper and Comma.ai's steering angle prediction model mentioned above were good referrence on this topic.

My first step was to use a convolution neural network model similar to the Nvidia's model, I thought this model might be appropriate because they have succeeded on real 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 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 by using some dropout layers between fully connected layers.

The final step was to run the simulator to see how well the car was driving around both track one and track two. I found there was a left turn bias on track one, so I collected counter-clockwise laps data around the track one and used image flipping technique to overcome this obstacle.

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

2. Final Model Architecture

The final model architecture (model.py lines 124-145) consisted of a convolution neural network with the following layers and layer sizes.

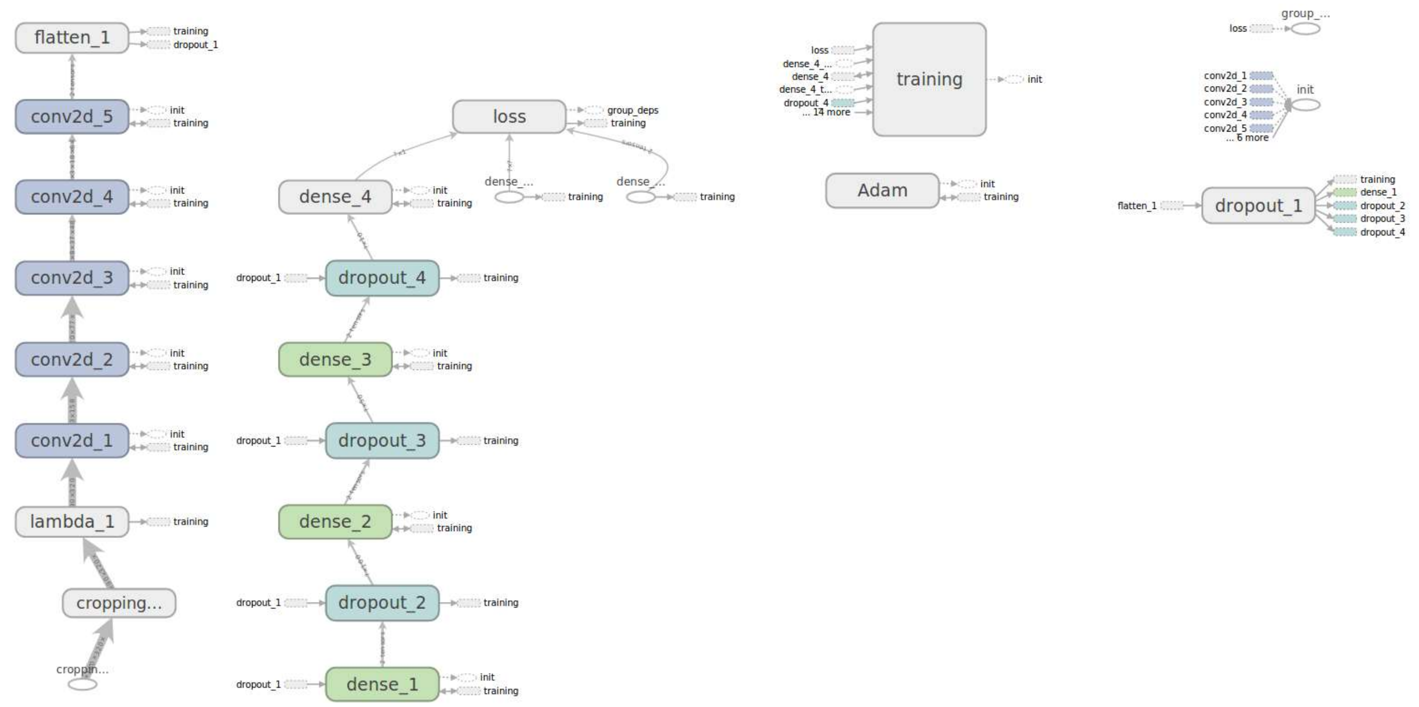
```
model = Sequential()
model.add(Cropping2D(cropping=((50, 20), (0, 0)),input_shape=(160, 320, 3)))
model.add(Lambda(lambda x: x / 127.5 - 1.0))
model.add(Conv2D(24, (5, 5), strides=(2, 2), activation='relu'))
model.add(Conv2D(36, (5, 5), strides=(2, 2), activation='relu'))
model.add(Conv2D(48, (5, 5), strides=(2, 2), activation='relu'))
model.add(Conv2D(64, (3, 3), strides=(2, 2), activation='relu'))
model.add(Conv2D(64, (3, 3), strides=(2, 2), activation='relu'))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(10, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))
```

The model's summary is following(output by Keras):

Layer (type)	Output Shape	Param #
cropping2d_1 (Cropping2D)	(None, 90, 320, 3)	0
Normalization (Lambda)	(None, 90, 320, 3)	0
conv2d_1 (Conv2D)	(None, 43, 158, 24)	1824
conv2d_2 (Conv2D)	(None, 20, 77, 36)	21636
conv2d_3 (Conv2D)	(None, 8, 37, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 18, 64)	27712
conv2d_5 (Conv2D)	(None, 1, 8, 64)	36928
flatten_1 (Flatten)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 100)	51300
dropout_2 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 50)	5050
dropout_3 (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 10)	510
dropout_4 (Dropout)	(None, 10)	0
dense_4 (Dense)	(None, 1)	11

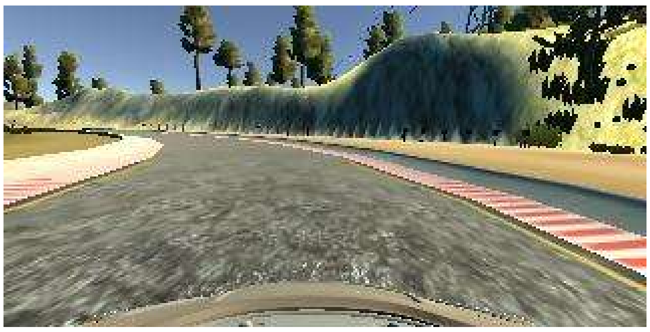
Total params: 188,219 ; Trainable params: 188,219 ; Non-trainable params: 0

Here is a visualization of the model architecture generated by Google's TensorBoard library (note: visualizing the architecture is optional according to the project rubric).

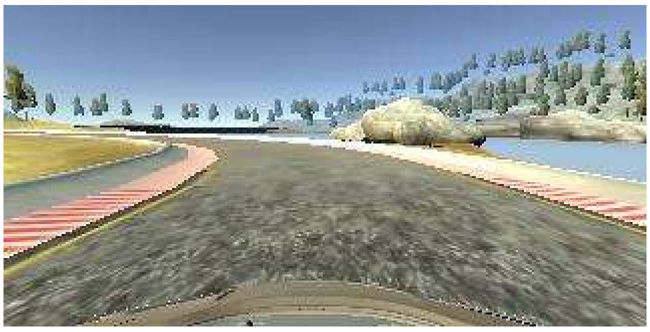


3. Creation of the Training Set & Training Process

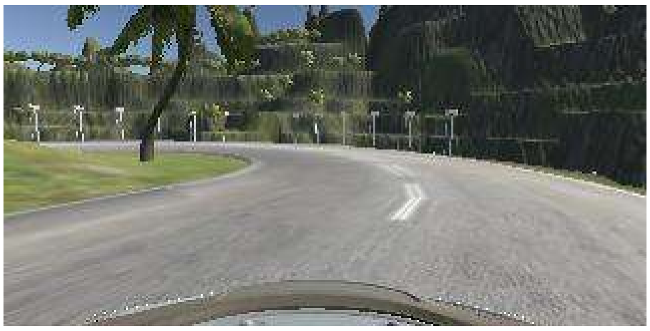
To capture good driving behavior, I first recorded three laps on track one using center lane driving. Here is an example image of center lane driving:



I then recorded four laps driving in a counter-clockwise direction on track one. Because track one has a left turn bias, if I only drive around the first track in a clockwise direction, the data will be biased towards left turns. Driving counter-clockwise is one way to combat the bias and is also like giving the model a new track to learn from, so the model will generalize better. Here is an example image of driving counter-clockwise:



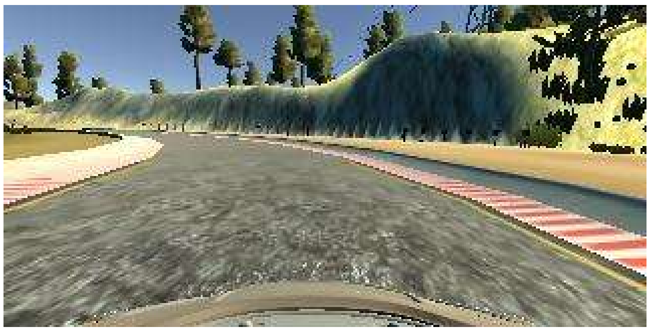
Then I repeated this process on track two in order to get more data points to generalize the neural network model better. Here is an example image of center lane driving in track two:



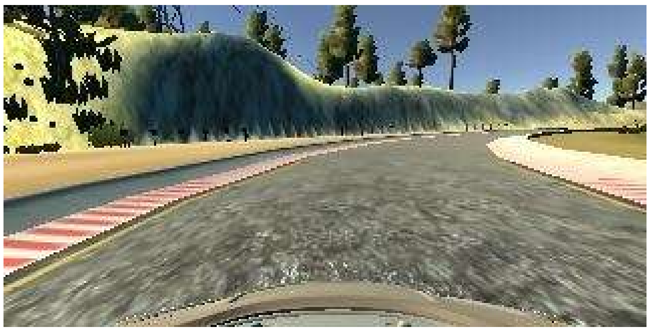
To augment the data sat, I also flipped images and angles thinking that this would help with the left turn bias. For example, here is an image that has then been flipped(using numpy.fliplr() function to do this, model.py lines 94-107):

```
import numpy as np
image_flipped = np.fliplr(image_original)
steering_angle_flipped = -steering_angle
```

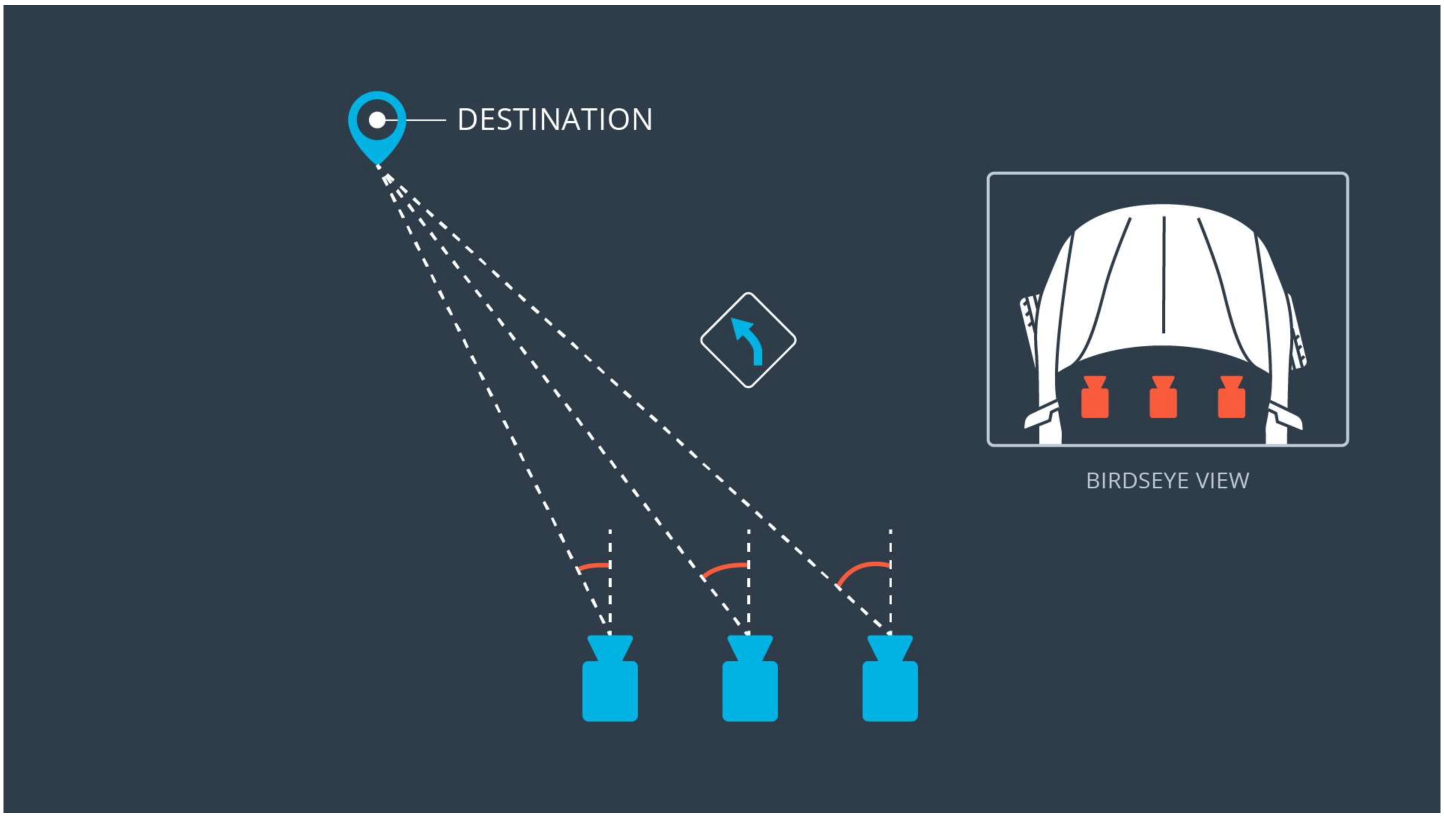
Original Image Example



Flipped Image Example



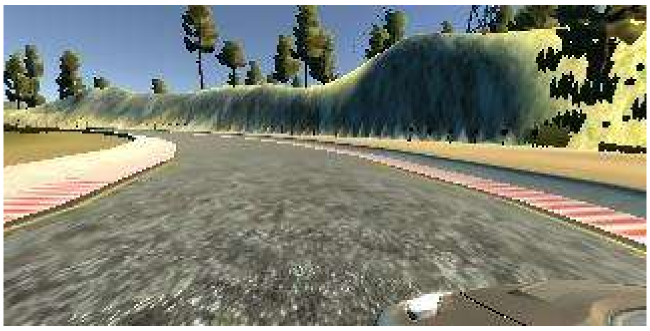
I also used multiple cameras' images. The simulator captures images from three cameras mounted on the car: a center, right and left camera. The following image shows a bird's-eye perspective of the car. From the perspective of the left camera, the steering angle would be less than the steering angle from the center camera. And from the right camera's perspective, the steering angle would be larger than the angle from the center camera(model.py lines 71-90).



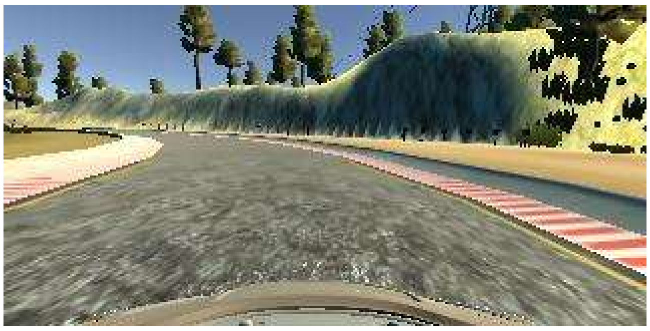
I chose `steering_correction=0.2` , and used following code segments to calculate multiple cameras' steering angles.

```
left_angle = center_angle + steering_correction
right_angle = center_angle - steering_correction
```

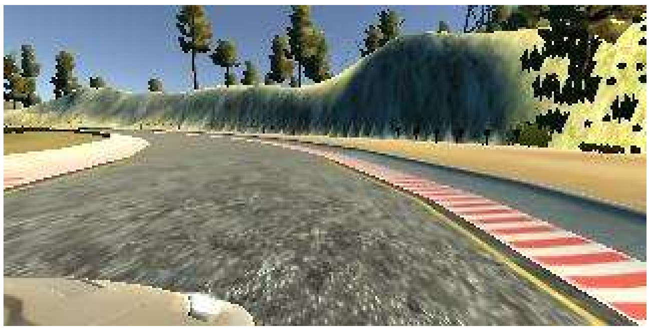
Left Camera Image Example



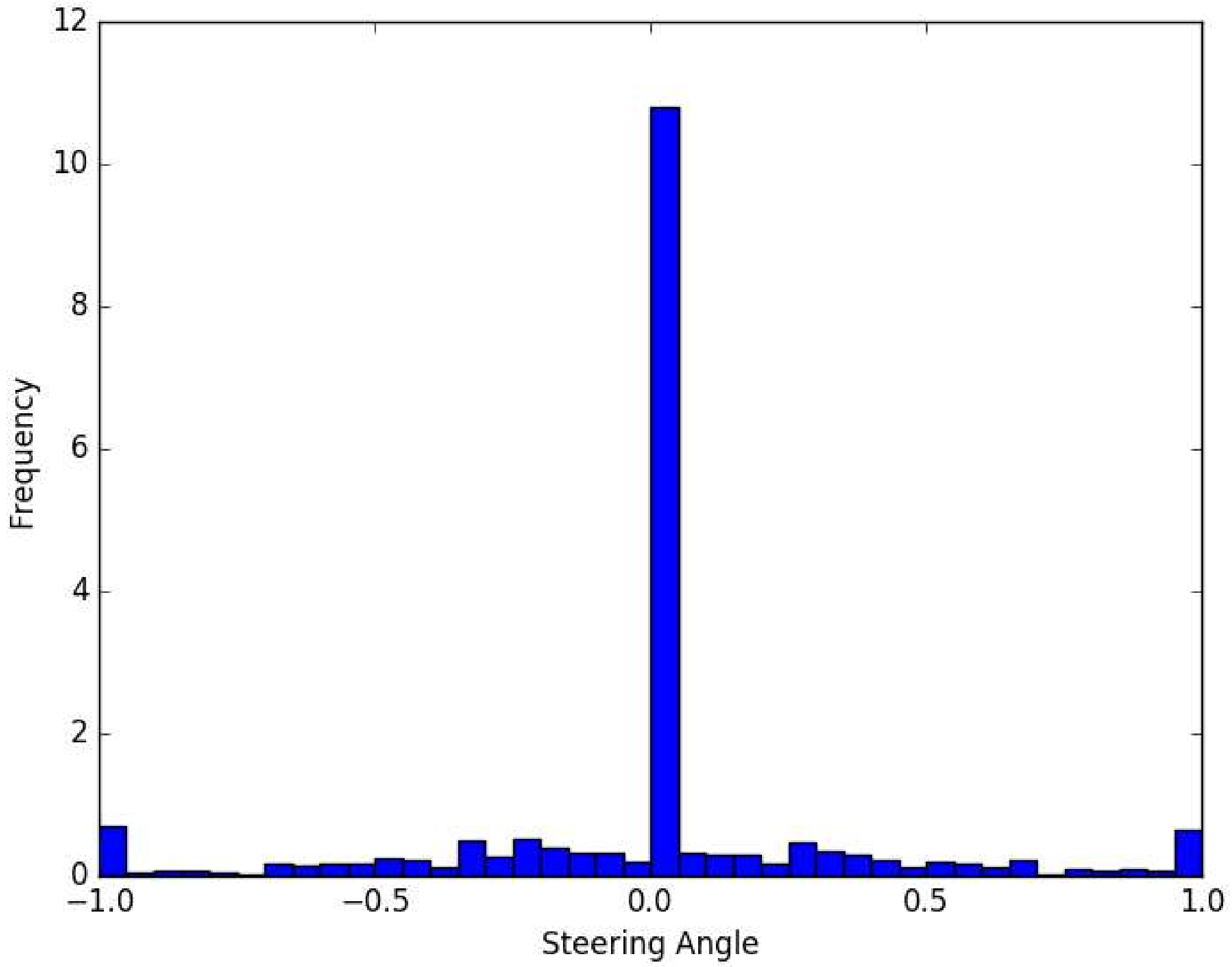
Central Camera Image Example



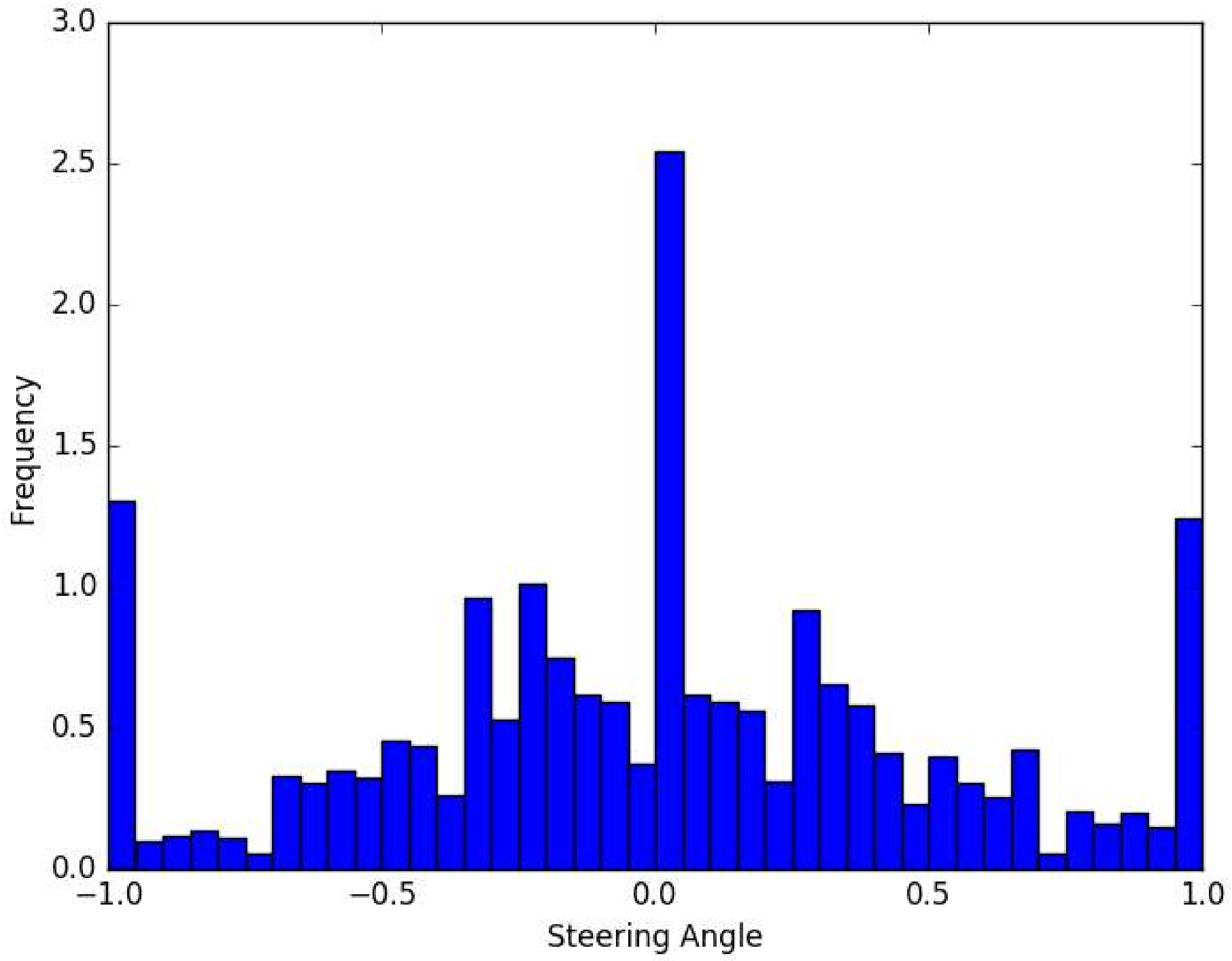
Right Camera Image Example



After the collection process, I had 14,937 track one driving data samples and 30,184 track two data samples(before using multiple cameras and image flipping technique to augment the driving dataset). The distribution of steering angles is shown below:

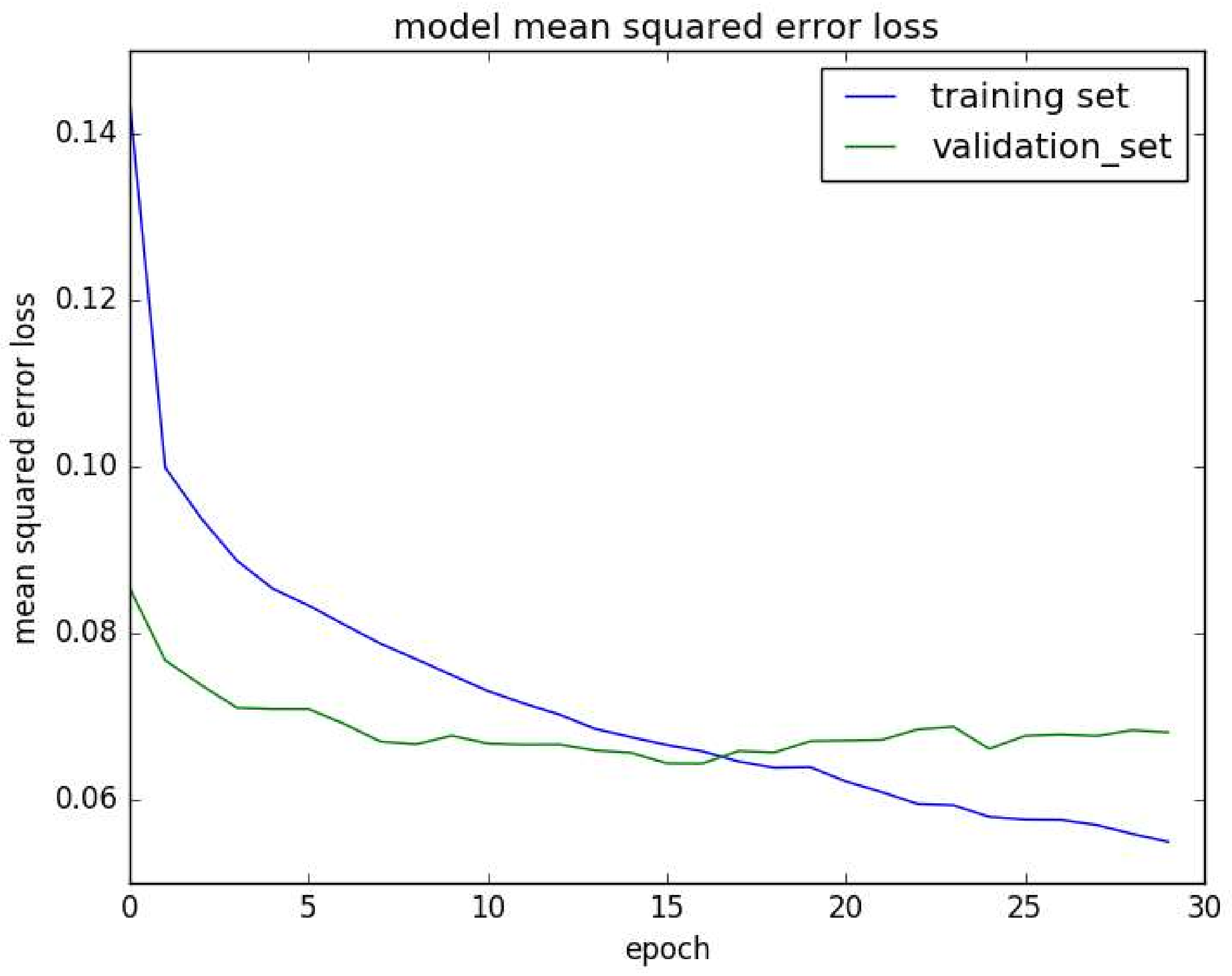


From the figure above, we can figure out the the driving data samples with 0 degree steering angle were overrepresented, and the dataset is too imbalanced. So I filtered about 90% driving data samples with 0 degree steering angle(model.py lines 32-39). After that, the distribution of steering angles is shown below:



I finally randomly shuffled the data set and put 20% of the data into a validation set and use generator function to feed data into the model.(model.py lines 166-169)

I used this training data for training the model. The validation set helped determine if the model was overfitting or underfitting. The ideal number of epochs was 16 as evidenced by the following image which showed the training history. I used an adam optimizer so that manually training the learning rate wasn't necessary.


model mean squared error loss
training set
validation_set
mean squared error loss
0.14
0.12
0.10
0.08
0.06
epoch
0
5
10
15
20
25
30