Behavioral Cloning

Image processing

1. Cropping

To reduce irrelevant image info, I cropped the top 70 pixels and bottom 25 pixels for each input image. This turn original 160x320 image into 65x320.

2. Normalization

I normalize all pixel value to be between [-1, 1] with a mean value of 0.

Model Training

1. Model Architecture

I used the NVidia's end to end training network presented in their paper. This network consists of 5 convolutional layers and 5 fully connected layers.

Here's the model details:

- Convolution layer. 24 filter of size 5x5, stride size 2x2. No padding. Input shape (None, 65, 320, 3). Output shape (None, 31, 158, 24)
- Covolution layer. 36 filter of size 5x5, stride size 2x2. No padding. Input shape (None, 31, 158, 24). Output shape (None, 14, 77, 36)
- Covolution layer. 48 filter of size 5x5, stride size 2x2. No padding. Input shape (None, 14, 77, 36). Output shape (None, 5, 37, 48)

- Covolution layer. 64 filter of size 3x3, string size 1x1. No padding. Input shape (None, 5, 37, 48). Output shape (None, 3, 35, 64)
- Convolution layer. 64 filter of size 3x3, stride size 1x1. No padding. Input shape (None, 3, 35, 64). Output shape (None, 1, 33, 64)
- Flatten layer. Input shape (None, 1, 33, 64). Output shape (None, 2112)
- Fully connected layer. 500 output nodes. Input shape (None, 2112). Output shape (None, 500)
- Fully connected layer. 100 output nodes. Input shape (None, 500). Output shape (None, 100)
- Fully connected layer. 50 output nodes. Input shape (None, 100). Output shape (None, 50)
- Fully connected layer. 10 output nodes. Input shape (None, 50). Output shape (None, 10)
- Fully connected layer. 1 output nodes. Input shape (None, 10). Output shape (None, 1)

I didn't use any activation layer nor dropout layer.

2. Training Data

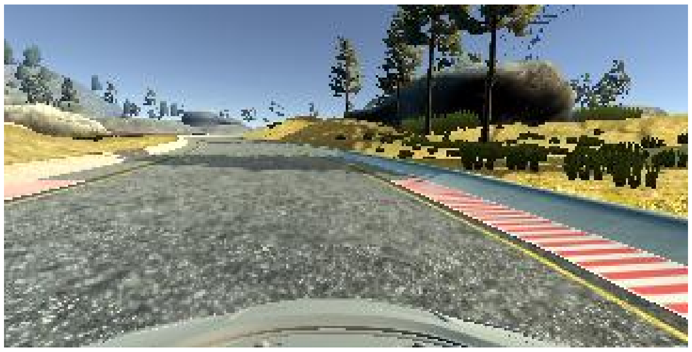
To increase the amount of training data, I trained the model from:

- Driving data provided by Udacity (downloaded from course website)
- Driving log on track 1
- Driving log on track 1 (reverse-direction)
- Two driving logs for steering from edge of the road to the center
- Driving log on track 2
- Driving log on track 2 (reverse-direction)

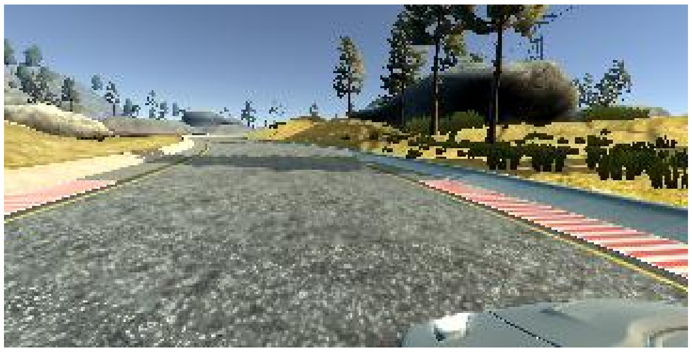
To increase the accuracy for recording steering angle, I use my PS3 analog stick to control the car. It does make a huge difference in terms of the data quality!

Besides I also use the left and right camera (with opposite forced steering offsets). I also flipped the image horizontally and use inverted steering angles:

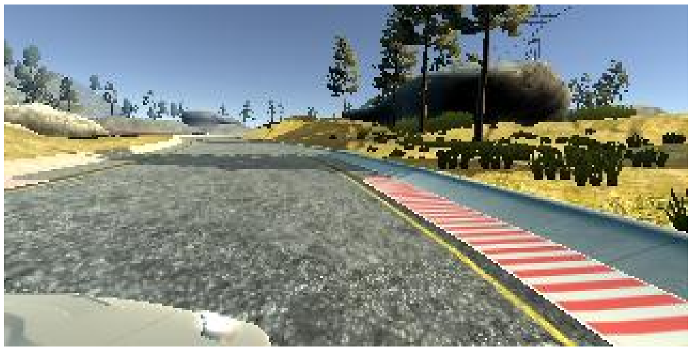
Center camera. Steering angle -0.276604.



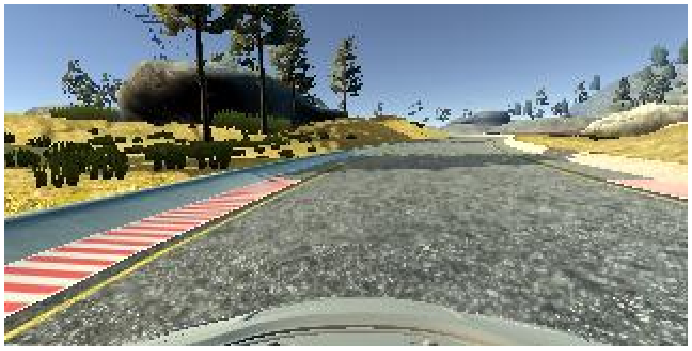
Left camera. Steering angle -0.276604 + 0.5 = 0.223396.



Right camera. Steering angle -0.276604 - 0.5 = -0.776604.



Flipped center camera. Steering angle = -(-0.276604) = 0.276604



Therefore, one driving log could provide 4x training data.

3. Model Parameter Tuning

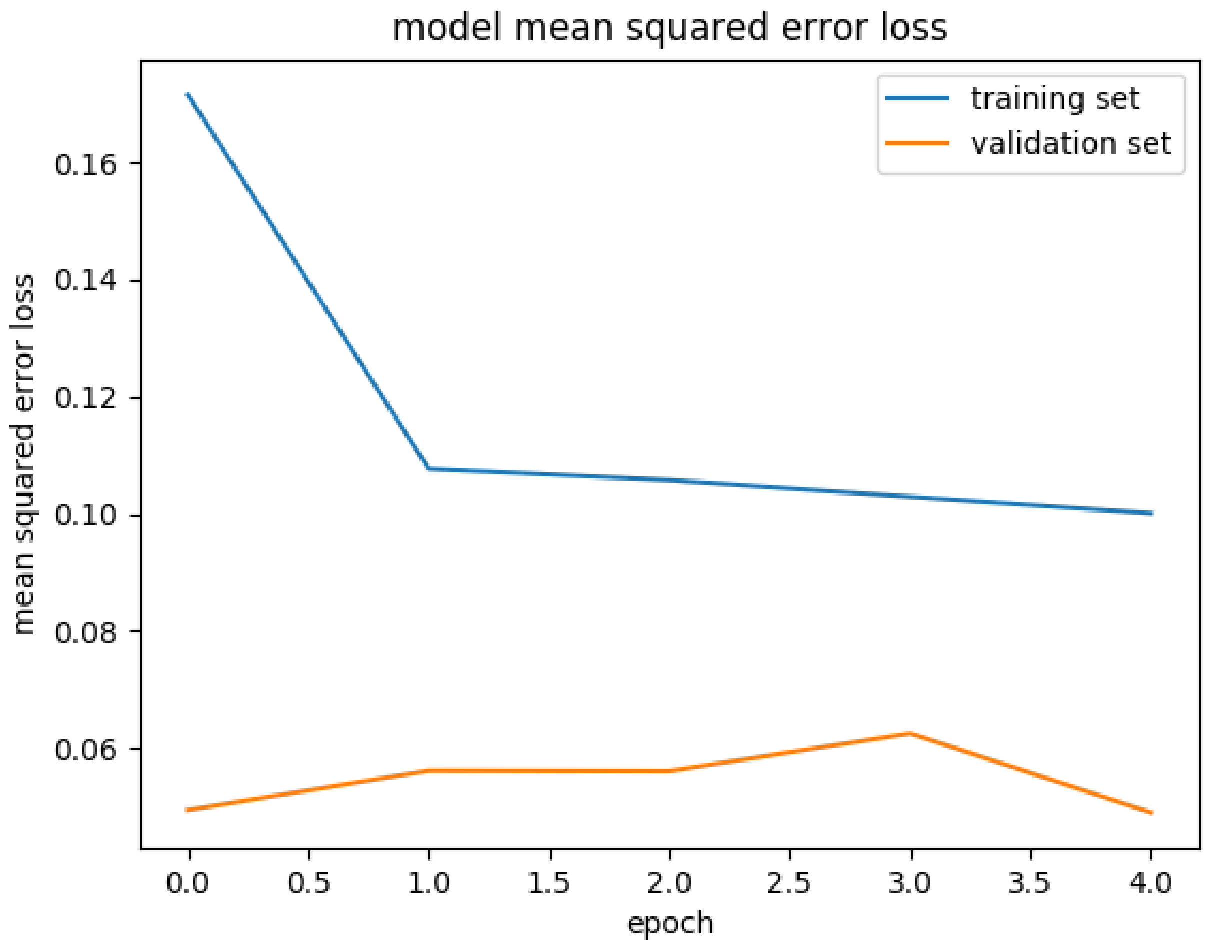
The model used an adam optimizer, so the learning rate was not tuned manually.

For batch size and number of epoch, since my GPU only has less than 4GB display memory, I have to set the batch size to be smaller (64). I watched the training and realize that after a few epochs the loss won't really get lower so I decided to stop training after 5 epochs.

Another interesting parameter to tune is the steering offset when using left or right camera images. Based on my experience, if I set it too high (such as > 1.0), the model will learn how to handle edge quickly, but it also becomes too sensitive to lane direction changes so the car will keep wobbling even on a straight lane. If I set it too low (such as < 0.2), the car cannot handle sharp turn, especially when the speed is set higher (such as 20mph). In the end, I choose 0.5 as it gives me good reaction while not being too sensitive.

Model Performance

Here's the training/validation loss for my training process:



When I tried my model for automonous driving, the car was able to finish driving track 1 at 20mph without leaving the lane. It could also finish driving track2 but with only 10mph. Please see my track1 driving video.