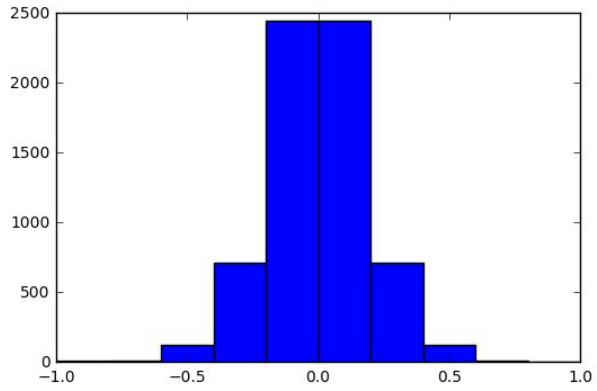
For this project, I first needed to decide how to load and process the images. I choose to use the Udacity dataset so that I could achieve better and more accurate results. I also collected about 4000 additional training images on the bridge and the final turn. I read in the images from the csv, and flipped the images vertically to double the amount of images I could work with. I also cropped 26 pixels off of the top of my images so that the scenery was cut out and the algorithm could focus solely on the road. I did this preprocessing for all of the center, left, and right images provided by Udacity. I then added in the steering values to map the images to. I split up the data into validation and training data, and shuffled the data to ensure that the sequence of the images wouldn't affect results.

I then split up the steering angle turns for the center images, using a threshold of 0.15 to append certain images to left and right turns. I did the same for the flipped center images, so I had lists of how the car turned from the center images. I then adjusted steering angle values for the left and right images to help add in recovery data and ensure that the steering angles were far from zero. I then put my straight steering images through a loop that weeded out images with steering angles of 0 so that the data wouldn't converge to a single angle. All of my angles hover around 0, but aren't exactly 0. The distribution of my data is attached as a file.

I then decided to follow the Nvidia Architecture, using 5 convolutional layers and three fully-connected layers. My input shape was (40,200,3) and my first convolutional layer had 1824 parameters and the shape (18, 98, 24). My second cone layer had the shape (7, 47, 36) and had 21636 parameters. The third conv layer had shape (2, 22, 48) and 43248 parameters. Finally, my last conv layer had shape (1, 31, 64) and 12352 parameters. I then flattened this conv layer and fed it into a fully connected layer with output of 100 and 134500 parameters. This fully connected layer fed into the second fully connected layer with shape 50, and 5050 parameters, finally feeding into the fully connected layer with output of 10, and then the last output layer with a shape of 1, to be the steering angle. I used the same kernel and stride sizes as the Nvidia Architecture. I decided to use an adam optimizer, and reduced the learning rate to prevent overfitting. I wanted to train for only 12 epochs to give the model enough time to learn, but not to overfit the data. I received solid results from the output of my architecture in terms of loss.

Images:



Histogram of my steering angles

Some images from the dataset: