**About this project**

This repository contains code for a project I did as a part of [Udacity's Self Driving Car Nano Degree Program](https://www.udacity.com/drive). We had to train a car to drive itself in a video game. The car was trained to drive itself using a deep neural network. Scroll down to see the demo video.

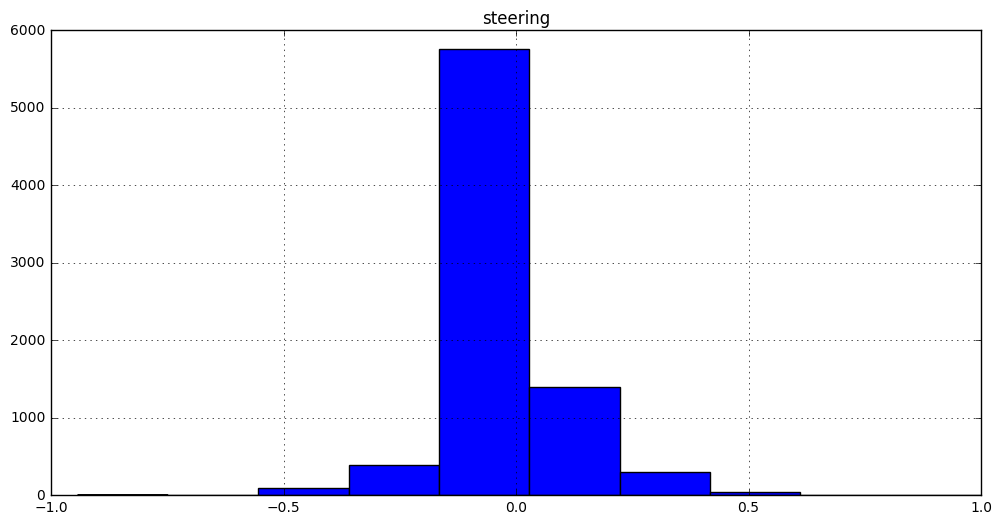
**Exploring the Dataset**

I used the dataset provided by Udacity. About 8000 images. I did not record images myself.

The dataset contains JPG images of dimensions 160x320x3. Here are some sample images from the dataset.

[](https://github.com/subodh-malgonde/behavioral-cloning/blob/master/assets/sample_images.png)

**Unbalanced Data**

Most of the steering angles are close to zero, on the negative side. There is a bias towards driving straight and turning left. [](https://github.com/subodh-malgonde/behavioral-cloning/blob/master/assets/unbalanced_data.png)

**Left/Right Camera ~= Parallel Transformation of the car**

The left and right cameras point straight, along the length of the car. So the left and right camera are like parallel transformations of the car.

[](https://github.com/subodh-malgonde/behavioral-cloning/blob/master/assets/cameras.png)**Use left & right camera images to simulate recovery**

Using left and right camera images to simulate the effect of car wandering off to the side, and recovering. We will add a small angle 0.2 to the left camera and subtract a small angle of 0.2 from the right camera. The main idea being the left camera has to move right to get to center, and right camera has to move left.

**Flip the images horizontally**

Since the dataset has a lot more images with the car turning left than right(because there are more left turns in the track), you can flip the image horizontally to simulate turing right and also reverse the corressponding steering angle.

**Preproceesing Images**

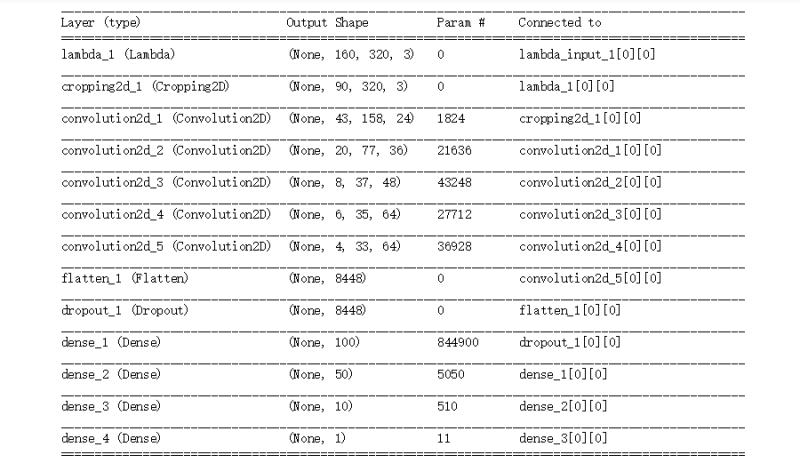
1. I noticed that the hood of the car is visible in the lower portion of the image. We can remove this.
2. Also the portion above the horizon (where the road ends) can also be ignored.

After trial and error I figured out that shaving 55 pixels from the top and 25 pixels from the bottom works well.

**Data Generation Techniques Used**

Data is augmented and generated on the fly using python generators. So for every epoch, the optimizer practically sees a new and augmented data set.

**Model Architecture**



1. **Layer 1**: use lambda to normalize the picture;
2. **Layer 2**: use cropping to cut some useless part of pictures
3. **Layer 3**: Conv layer with 24 5x5 filters, ,subsample(2x2),RELU activation
4. **Layer 4**: Conv layer with 36 5x5 filters, ,subsample(2x2),RELU activation
5. **Layer 5**: Conv layer with 48 5x5 filters, ,subsample(2x2),RELU activation
6. **Layer 6**: Conv layer with 64 3x3 filters, RELU activation
7. **Layer 7**: Conv layer with 64 3x3 filters, RELU activation
8. **Layer 8**: flatten the image\_data
9. **Layer 9**: dropout rate 50%
10. **Layer10-13**: dense the output to one;

**Training Method**

1. Optimizer: Adam Optimizer
2. No. of epochs: 10
3. Training samples:19286
4. Validation Set: 4822

**Evaluation Video**

The output video is video.mp4.