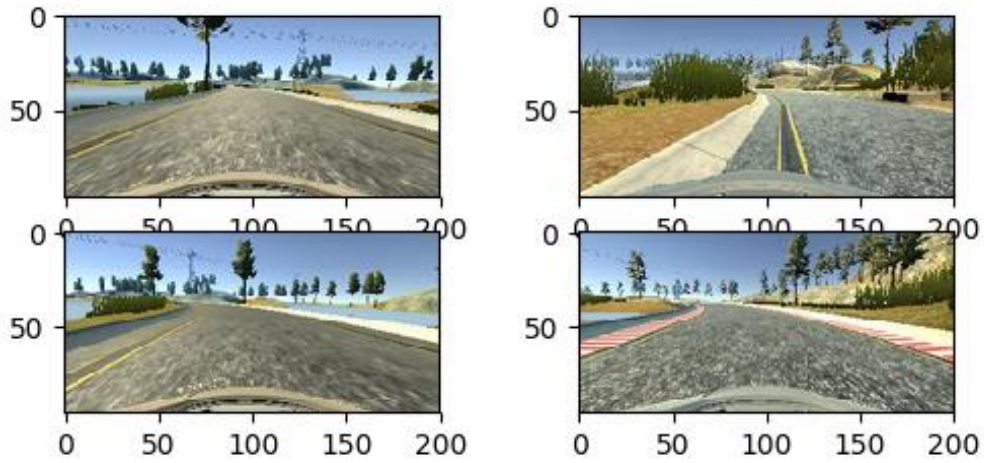
Behavioral Cloning Project

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 Kera's 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

Except for resizing of images for ease of training, no other preprocessing is done on the images. No image augmentation techniques like flipping etc was also not undertaken. As no heavy preprocessing and augmentation of images is undertaken, generators are not used in the code. (As I generated lot of data using different techniques, I felt there is no need for augmentation)

Picture of loaded images



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.pdf summarizing the results

Model Architecture and Training Strategy

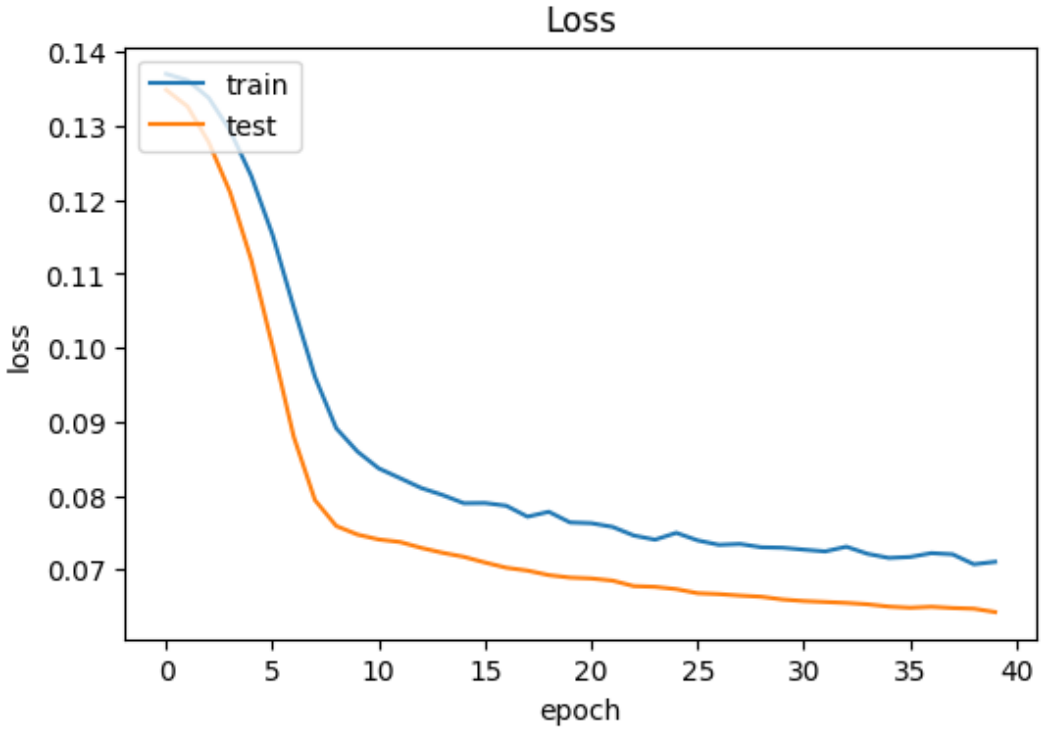
As NVIDIA architecture is proven and light, it is slightly modified and utilized for training. VGG16 was also tried out but because of its heavy architecture and performance is also not better than NVIDIA, finally it was decided not to use it. The model architecture is as follows: - (code from line 102 to 120 of model.py)

Layer	Details
Lambda layer	Normalization input size is (96,200,3)
Cropping layer	Cropping size ((25,5), (0,0))
Convolution layer	5x5 Filter size, stride 2x2 with a depth of 24 layers, valid padding and Relu activation
Convolution layer	5x5 Filter size, stride 2x2 with a depth of 36 layers, valid padding and Relu activation
Convolution layer	5x5 Filter size, stride 2x2 with a depth of 48 layers, valid padding and Relu activation
Convolution layer	3x3 Filter size with a depth of 64 layers, valid padding and Relu activation
Convolution layer	3x3 Filter size with a depth of 64 layers, valid padding and Relu activation
Flatten layer	Layer flattened out
Dropout layer	30% connections are dropped out
Dense layer	1164 neurons in this layer with Relu activation
Dropout layer	30% connections are dropped out
Dense layer	100 neurons in this layer with Relu activation
Dropout layer	30% connections are dropped out
Dense layer	50 neurons in this layer with Relu activation
Dropout layer	30% connections are dropped out
Dense layer	10 neurons in this layer with Relu activation
Dense layer	1 neuron in this layer with linear activation for predicting a single value for steering angle

Lot of dropout layers are introduced to avoid overfitting. Non-linearity is introduced by utilizing the Relu activation in hidden layers and in the final layer linear activation was used. Training and validation data sets are created separately and the model was trained and validated using different datasets (code line 139 and 142). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

The model used was Stochastic Gradient Descent. The learning rate varies according to the decay rate. In the initial stages, high learning rates will make major changes to the weight but as the epochs progresses the learning rate will decay to avoid missing the lowest point in the gradient. Momentum is also choosed to drive the weights in the proper direction. (code lines 124 to 129 of model.py)

First, I tried with very low epoch values. The training and validation loss was reducing for all epochs. So, the number of epochs was slowly increased gradually from 5 to 15,20,30 and finally 40. Through all these values the training and validation loss continued to decrease. I plotted the training and validation loss to decide if the epoch values should further be reduced.



The loss of both training and validation seems flattened out. Further increasing the epochs may further improve the loss by very small amounts. Considering the computational time and effort required and the model's performance in the simulator, it was decided not to further increase the number of epochs. The model was checkpointed to save the best weights during training. These weights are again loaded back into the program and utilized for calculating loss on test data. The checkpoint was created in such a way that the weights will get saved whenever the validation loss is improved from the previous best score.

Creation of Training Data

I used the data provided by Udacity along with my own generated data. Initially when I used only Udacity dataset the car was completely shifted towards the right of the road and was going away from the track at the very first turn. The car was not getting recovered back. To avoid this I generated my own data. I collected data by going around the track in clock and anti-clock wise direction for 2 times. This was done to keep the car in the middle of the road and avoid left and right bias. Next I generated data by doing a recovery lap. This data was merged with Udacity dataset and utilized for training.

Architecture and Training Documentation

First the training was undertaken with VGG16 architecture using only Udacity data. As the architecture is quite heavy it was taking lot of time for training. But the training and validation loss was both low. Then to improve the time taken to train the architecture was changed. As the NVIDIA architecture was experimented in real world scenario and was successful, as it was very light and use very less computational resource, I decided to go with it. In order the improve the performance in the simulator, data was generated locally. The optimizer first selected was Adam and with this optimizer NaN was generated in training and validation. I searched the internet to find the reason for generation of NaN values. The most probable reason seems to be the vanishing gradient problem. So I decided to go with SGD by selecting the learning rate, decay and momentum manually. This avoided generation of NaN values. Every time I ran the model both validation and test loss is very low. This suggested that the model is very strong and the data utilized for training needs to be appropriate. Creation of appropriate data by combining the Udacity and self-generated data the car could drive around the simulator very successfully.