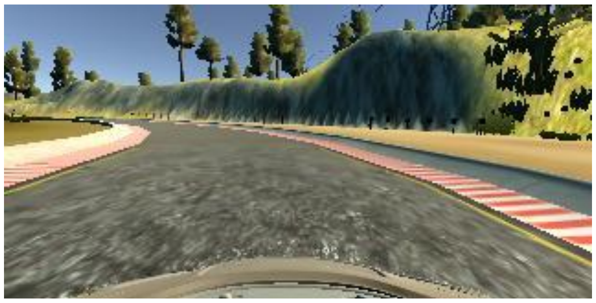
CarND Term1: Project 3: Behavioral Cloning

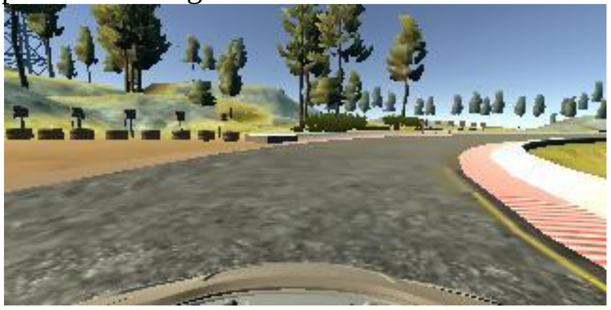
1. My project includes the following files:

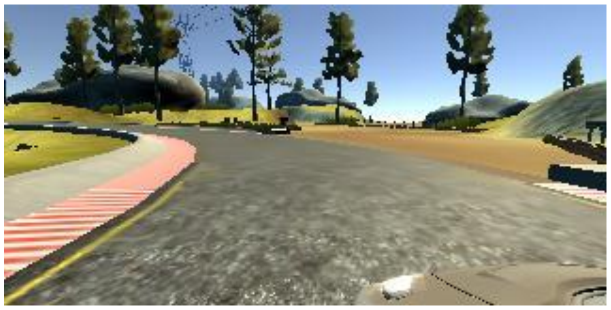
- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report.md or writeup_report.pdf summarizing the result
- video.mp4 - A video recording of the vehicle driving autonomously around the first track.
- video_second-track - A video recording of the vehicle driving autonomously around the second track (partial).

2. Choosing the right dataset:

- Started training the network with the data provided by Udacity but a quick look at the data indicated a heavy bias towards driving straight without any steering angle. This made it difficult for the car to navigate through the turns. So, collected three additional sets of data(mostly around the corners as shown below): one counter-clockwise lap, one clockwise lap and one around a particular corner right after the bridge.

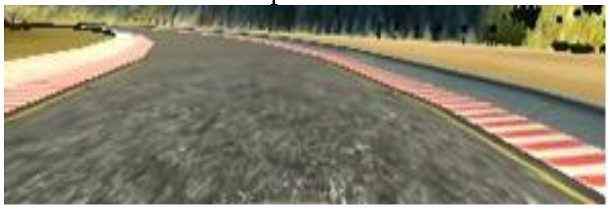

Set1: Example Turn during a Counterclockwise drive around the track


Set2: Example Turn during a Clockwise drive around the track


Set3: CCW turn after the bridge

3. Pre-processing the data:

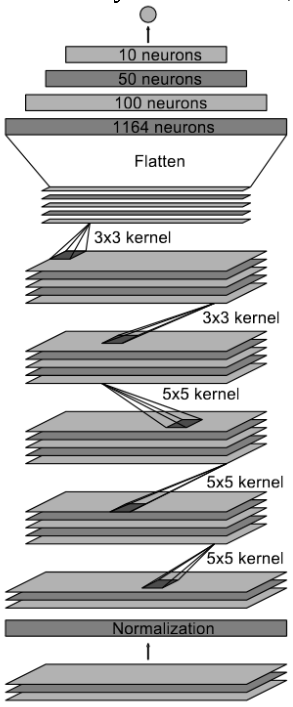
- Normalized the data by dividing with 255 (as the input being RGB images) and subtracted 0.5 to mean-center the data to 0. This step helps the network quickly converge to a solution and avoid dealing with large numbers.
- Cropped the top 70(hills, trees, sky) and bottom 25(hood of the car) rows of pixels that do not have information of the track. This helps the network concentrate mostly on the track.



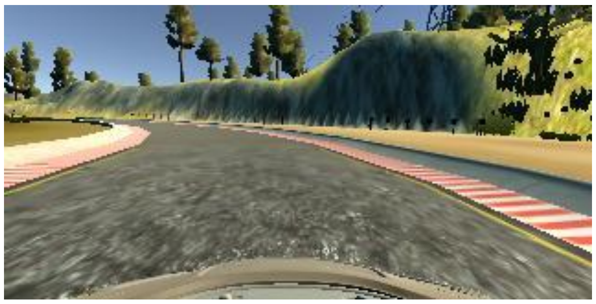
CCW driving example image from above after cropping

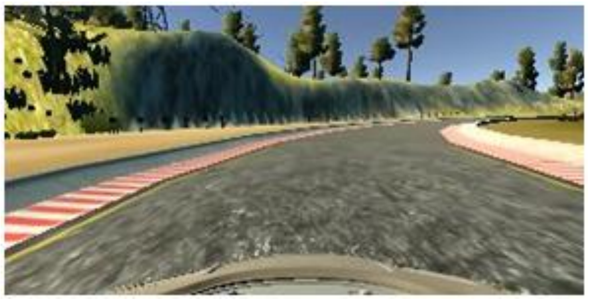
4. Model Architecture:

- I used the NVIDIA pipeline(from this source) described in Lesson 14 of the Behavioral Cloning module. The pipeline below taken from the published paper in the link above visualizes the network. The final model architecture (written in the keras_model class in the model.py python code) consisted of the following layers in this order:
    i. Convolution layer with 24 5x5 filters followed by a Relu activation
    ii. Convolution layer with 36 5x5 filters followed by a Relu activation
    iii. Convolution layer with 48 5x5 filters followed by a Relu activation
    iv. Convolution layer with 64 3x3 filters followed by a Relu activation
    v. Convolution layer with 64 3x3 filters followed by a Relu activation
    vi. Flatten
    vii. Three fully connected layers with 100, 50 and 10 neurons respectively.



- Used train and validation splits to divide the data in the ratio of 0.8:0.2.
  Started training initially with 7 epochs but later reduced it to 3 epochs (early termination) as the validation loss didn't change much after 3 epochs.
- Both the training loss and validation loss were low and <u>similar</u> during all three epochs. Also, the loss in both cases kept decreasing during the three epochs. Because of the above two reasons no additional measures were necessary to avoid overfitting, because of which I didn't have to use any other Regularization or Dropout techniques.
- Objective function was set to minimizing the Mean Squared Error(MSE) of the final steering output. Used Adam optimizer for optimization.
- Used generators to make the network memory efficient.
- I let the network randomly choose between center, left and right images to make the learning more robust. Used the following function to supply the steering correction for left and right images. steering_correction = 0.25*(3 - 2*i) if i else 0, it outputs 0 for i=0, 0.25 for i=1(left image) and -0.25 for i=2 (right image)
- Flipped selected images and took the negative value of their corresponding steering angle to augment training data. <u>Augmented only the images with a non-zero steering angle to compensate for the bias towards straight driving in the driving data.</u>


Original Image


Flipped Image

5. <u>Results</u>:

- As I improved the neural network, the following actions contributed to a significant improvement of the results:
  b) Using the NVIDIA pipeline with generators.
  c) Cropping and normalizing the images.
  d) Collecting more data around the corners.
  e) Randomly choosing between Center, Left and Right images
  f) Augmenting only the images with a non-zero steering angle.
- Even though I didn't use any training data from the second track, the model performed decently on the second track and I've included a video of the second track as well.

Thanks to Udacity discussion forums which helped me take care of many practical issues such as dealing with AWS problems and using generators.