BEHAVIORAL CLONING

File Submission/Code Quality

- **<u>Files Submitted:</u>**

  My project includes the following files:

  - model.py containing the script to create and train the model.
  - augment.py containing the script to augment the dataset.
  - drive.py for driving the car in autonomous mode
  - model.h5 containing a trained convolution neural network
  - writeup_report.pdf summarizing the results

- **<u>Submission includes functional code:</u>**

  Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing - $python\ drive.py\ model.h5$

- **<u>Submission code is usable and readable:</u>**

  The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works

Model Architecture & Training Strategy

- **<u>An appropriate model architecture has been employed:</u>**

  - I have used the NVIDIA Architecture for my model.
  - The model consists of normalization and cropping layers at the start which are Keras Lambda Layers (code lines 163, 164), then it consists of 3 Convolution layers with 2x2 stride and 5x5 kernel with 3, 24, 36 filters respectively.
  - Then the model is connected to two Convolution Layers with 1x1 stride and 3x3 kernel with 48 and 64 filters respectively
  - The model uses ELU Activation layers to introduce nonlinearity.
  - The model is flattened and is then connected to 5 Fully Connected Layers with 1164, 100, 50, 10 and 1 nodes.

- **<u>Attempts to reduce overfitting in the model</u>**
  - Dropout Layers have been added between the Fully Connected Layers to reduce overfitting (code line 177)

- The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 162-188). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

- **Model parameter tuning:**
  - The model used an adam optimizer, so the learning rate was not tuned manually

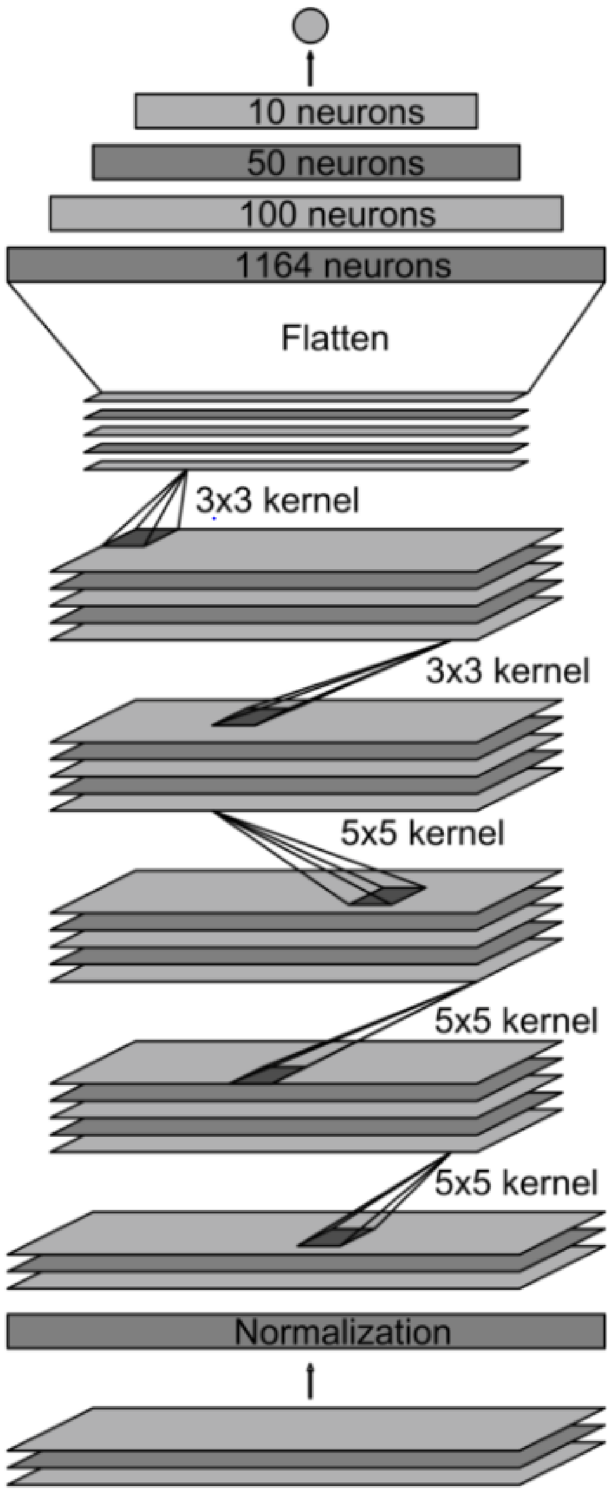
- **Appropriate training data:**
  - Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving for 2 laps, recovering from the left and right sides of the road for 1 lap, driving in reverse for 1 lap.
  - I also generated additional training data using image processing which will be explained in the next section.

Architecture & Training Documentation

- **Solution Design Approach:**

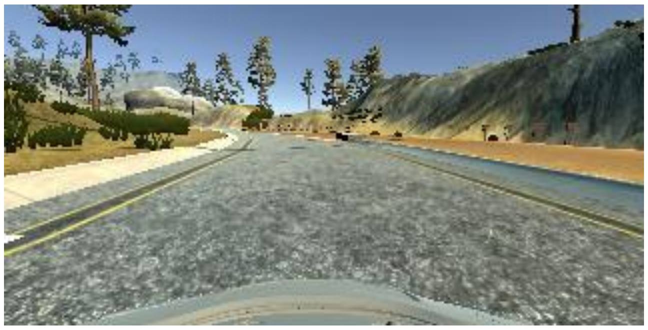
  - My first step was to use a convolution neural network model like the NVIDIA model. I thought this model might be appropriate because the NVIDIA model has performed well for the case of Self-Driving Cars and I thought it would be a perfect fit for the application of behavioral cloning. The model also uses a similar setup with three cameras in left, center and right and records the steering angles. As the setup was pretty similar to the setup in the project and the results were promising. So I have chosen this model.
  - In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.
  - To combat the overfitting, I modified the model by adding Dropout layers and augmenting the dataset with more images which were generated on the fly during the training process using python generator. I also added more data by driving in reverse direction to help the model generalize better.
  - The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track, mainly near steeper curves. To improve the driving behavior in these cases, I added more data to the dataset by driving to the sides of the road and then recording the car coming backup to the center of the lane. I did this for one entire lap.
  - At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

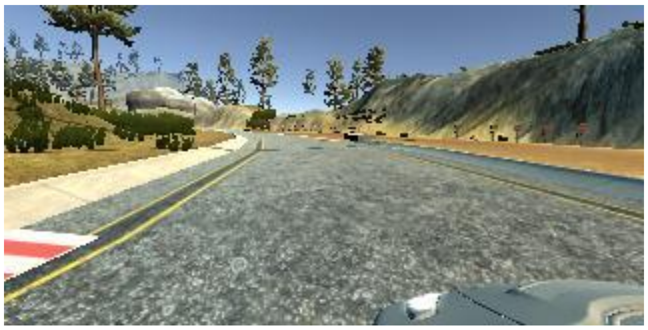
- **Final Model Architecture:**
  - The final model architecture (model.py lines 162-185) consisted of a convolution neural network with the following layers and layer sizes:
    - **Layer 1:** Keras Lambda Layer for Normalization
    - **Layer 2:** Keras Lambda Layer for cropping images to the Regions of Interest
    - **Layer 3:** Convolution Layer with 2x2 Stride size, 5x5 Kernel size and 3 Filters and ELU Activation Layer
    - **Layer 4:** Convolution Layer with 2x2 Stride size, 5x5 Kernel size and 24 Filters and ELU Activation Layer
    - **Layer 5:** Convolution Layer with 2x2 Stride size, 5x5 Kernel size and 36 Filters and ELU Activation Layer
    - **Layer 6:** Convolution Layer with 1x1 Stride size, 3x3 Kernel size and 48 Filters and ELU Activation Layer
    - **Layer 7:** Convolution Layer with 1x1 Stride size, 3x3 Kernel size and 64 Filters and ELU Activation Layer
    - **Layer 8:** Fully Connected Dense Layer with 1164 Nodes with Dropout Layer with 0.5 Drop Probability
    - **Layer 9:** Fully Connected Dense Layer with 100 Nodes with Dropout Layer with 0.5 Drop Probability
    - **Layer 10:** Fully Connected Dense Layer with 50 Nodes with Dropout Layer with 0.5 Drop Probability
    - **Layer 11:** Fully Connected Dense Layer with 10 Nodes with Dropout Layer with 0.5 Drop Probability
    - **Layer 12:** Fully Connected Dense Layer with 1 output Node
  - Here is a visualization of the architecture

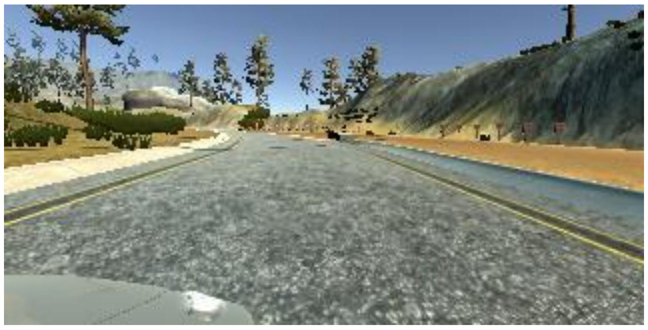


Reference - https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/

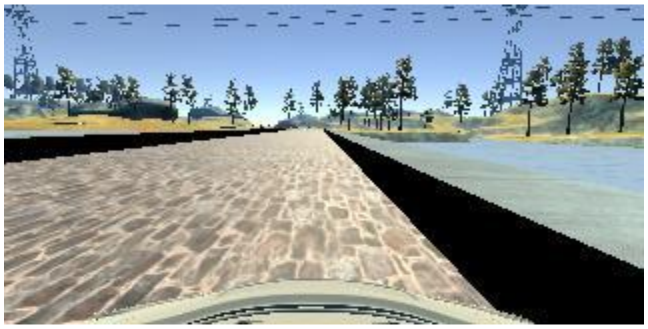
- **Creation of the Training Set & Training Process**
  - To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving from the center, left and right cameras respectively.

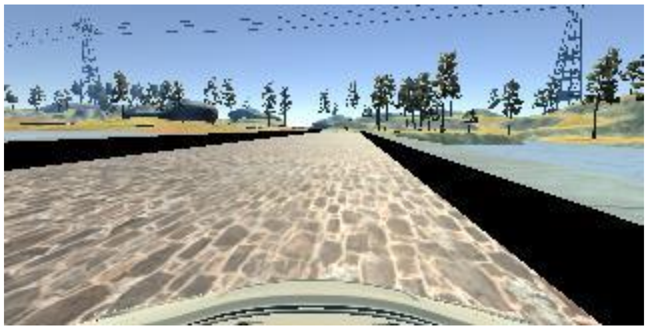


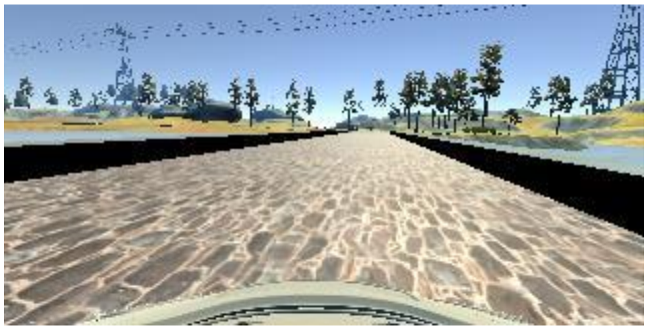




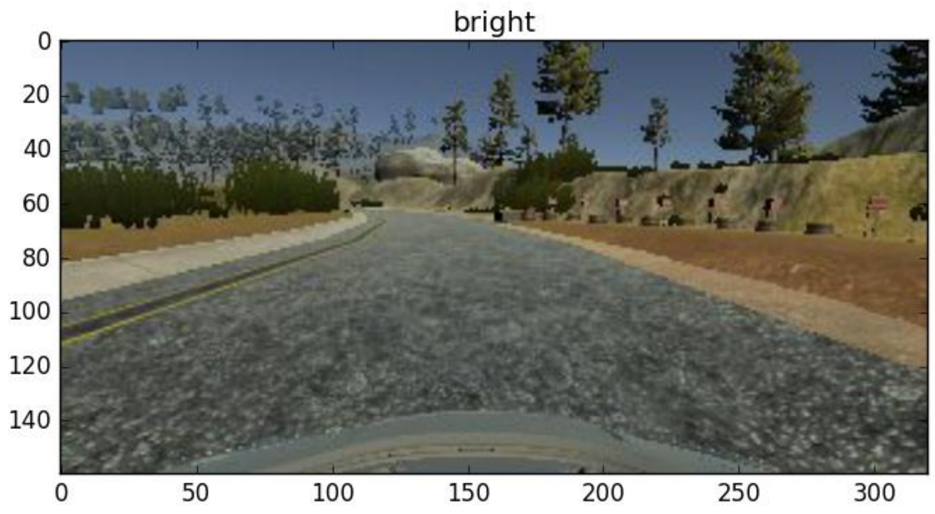
  - I then recorded the one lap with the car driving in reverse direction of the lap.
  - I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to get back to the center of the road, in case it starts moving away from the center and starts going off to the sides. These images show what a recovery looks like:

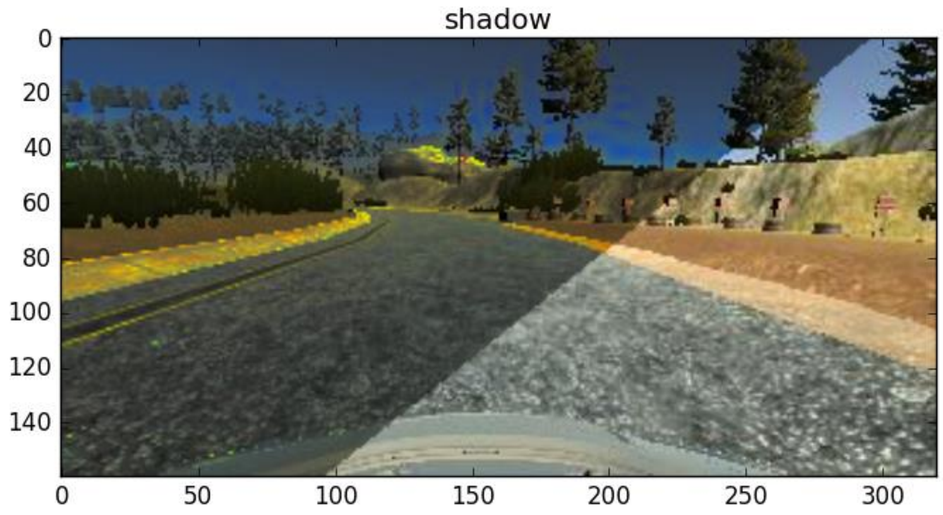


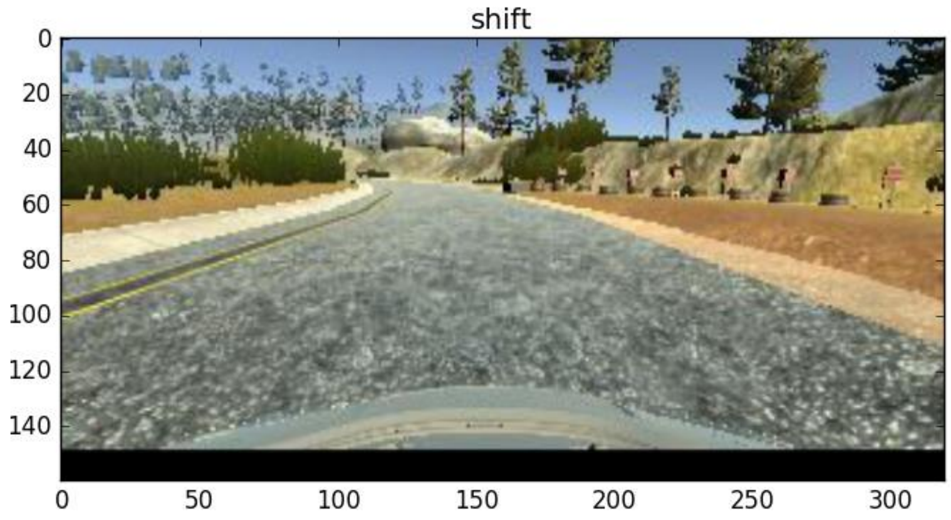


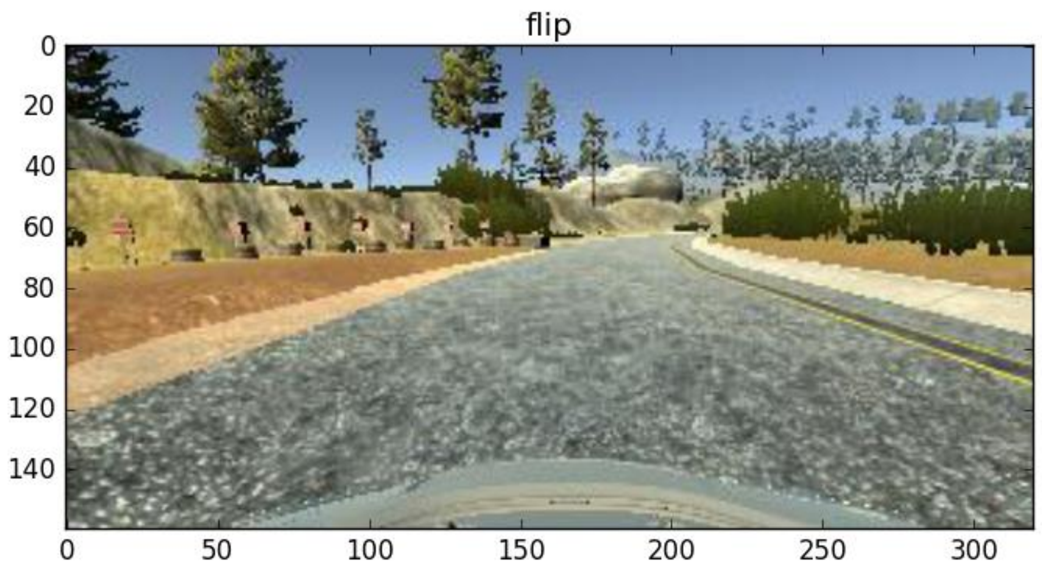


- **To augment the data set**:
- I flipped the images to make the model generalize better. I then augmented the dataset by changing the brightness of the images randomly by making the images brighter and darker as I thought this would help the model generalize various light conditions. I then augmented the data with shadows to help the model learn better in places with shadows.
- I also fed in the left and right camera images and made them look as if they were coming from center by adding a correction factor of 0.25 to the steering angles. This way, I though I can teach the model how to steer if the car drifts off to the left or the right. I also added some vertical shifts to the dataset.
- I also added some vertical and horizontal shifts to the dataset.
- The Brightness, Shadow and Flip augmentation process was done 3 times each for a single image.
- Here are some examples of images generated during the augmentation process along with their titles describing the operations:

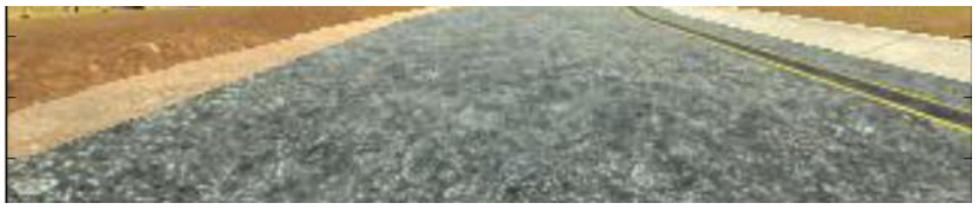

bright
0
20
40
60
80
100
120
140
0
50
100
150
200
250
300


shadow
0
20
40
60
80
100
120
140
0
50
100
150
200
250
300


shift
0
20
40
60
80
100
120
140
0
50
100
150
200
250
300



- After the collection process, I had 4400 number of data points plus 3 more points for each image which are generate in the augmentation process via python generator. I then preprocessed this data by normalizing it then cropping it to the regions of interest.
- Here is an example of image cropped to ROI:



- I finally randomly shuffled the data set and put 20% of the data into a validation set.
- I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 3 as the validation error started to increase after the third epoch so I decided to stop training after 3 epochs. I used an adam optimizer so that manually training the learning rate wasn't necessary.