Behavioral Cloning

Project overview:

The goal of the project is to use the simulator to capture human driving behavioral and teach the car to drive by itself.

The steps of the project are listed below:

1. Use the simulator to collect data which includes images, angles, speed etc.
2. Build a convolution neural network in Keras
3. Train and validate the model
4. Test on the autonomous mode to see if the car can correctly predict human driving behavior

File Submitted & Code Quality

My project includes the following files:

1. Model.py is a python script that is used to build, train, and validate the convolutional neural network
2. Drive.py is a python script that is used to driving the car in autonomous mode
3. Model.h5 contained a trained convolution neural network
4. Final report

The model.py file contains code for training and saving the convolutional neural network. The code is readable since comments are added to explain the models. In addition, the code is reusable since a generator is used to improve memory efficiency.

Model Architecture and training strategy

1. An appropriate model architecture has been employed

   My model consists of three 5x5 filter and two 3x3 filter. The filter depths are 24, 36, 48, 64, 64 respectively. The model also has one activation layer RELU for each convolutional layer. A lambda was added to the network to normalize the data and a cropping layer was added to the network to reduce distracting information.

2. Attempts to reduce overfitting in the model

   The model applied a dropout layer after each fully connected layer to reduce overfitting.

3. Model parameter tuning

   The model applied an adam optimizer, so the learning rate parameter did not need to be tuned.

4. Appropriate training data

   Training data was collected by always keeping the car driving on the center of the road. When we collect the data, we need to ensure that the car never leaves the track.

Model Architecture and Training Strategy

1. Solution Design Approach

The first model I built was the NVIDIA architecture because this model was published in a paper and was proven to be a powerful neural network.

I split the data into two sets: training data set and validation data set by a factor of 0.2. Then I used the generated training and validation data to see how well the model performed. At the beginning, I found out the training loss was much less than the validation loss. Therefore, I applied three dropout layers in between fully connected layers. I retrained the model, and I found out the validation loss was reduced.

Then I tested the built model on the simulator. The car fell off the track at the sharp turn right after passing the bridge.

To solve this problem, I decided to collect more training data by running more laps around the track.

Finally, the car was able to drive autonomously and safely on the track.

2. Final Model Architecture

The final model architecture I choose to apply is NIVDIA network which consists of three convolutional layers with 2x2 stride and a 5x5 kernel and a non-strided convolution with a 3x3 kernel size in the last two convolutional layers.

After these five convolutional layers, three fully connected layers and dropout layers were applied to the model.

The model architecture is shown below:

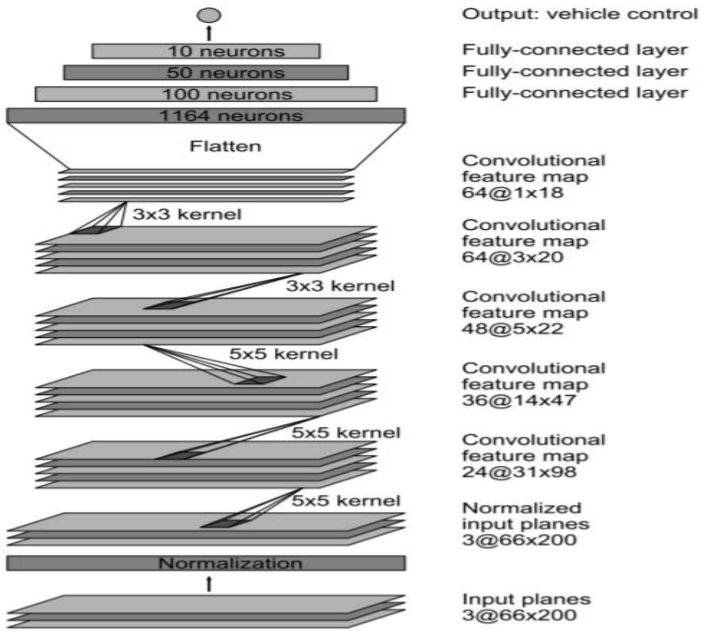
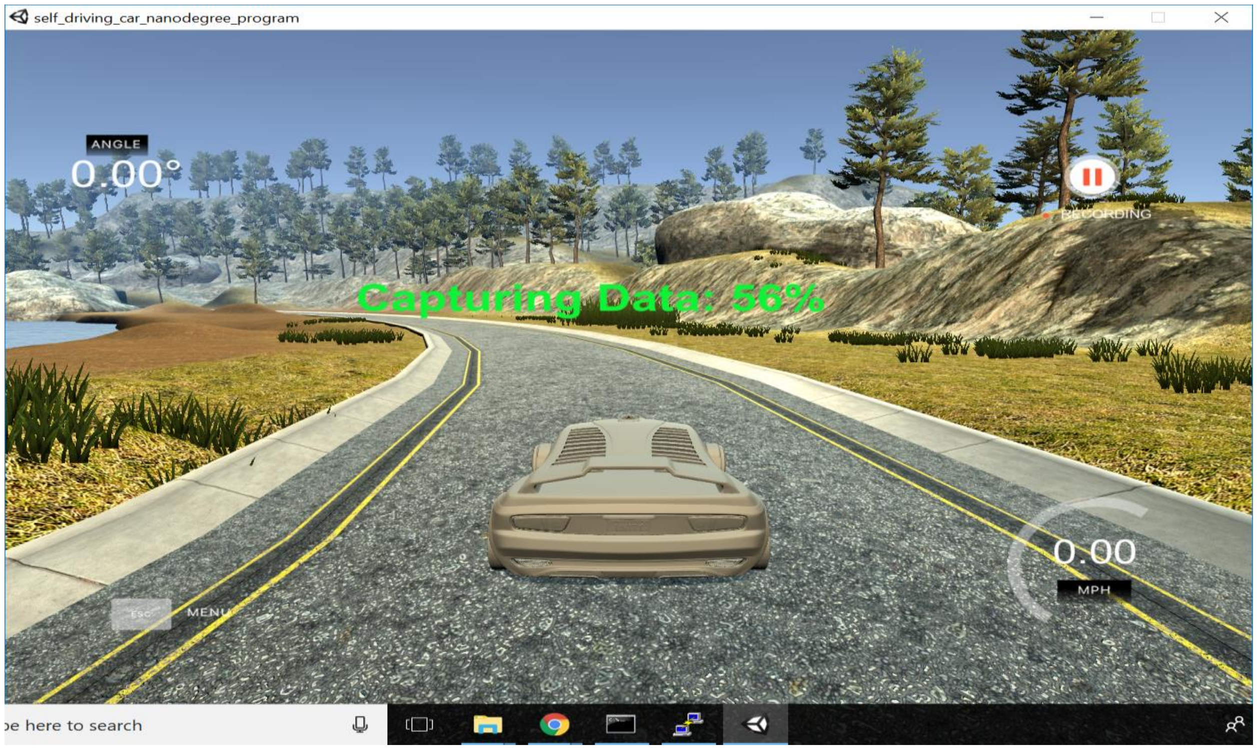


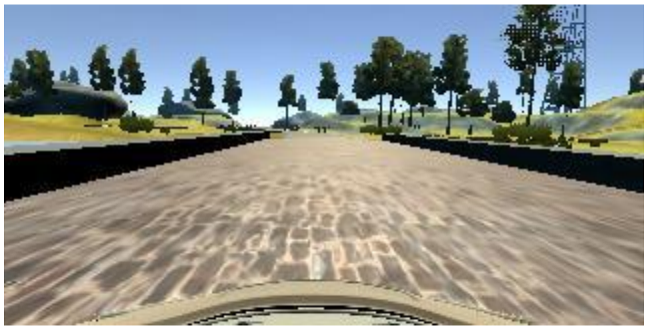
Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

3. Creation of the Training Set & Training Process
   I used the data set provided by Udacity. After I used this data set to train the model, I found out the car is not driving very well. So, I recorded at least two laps on track using center lane driving to gather more training data to help the model generalize better.
   The image below demonstrates center lane driving:

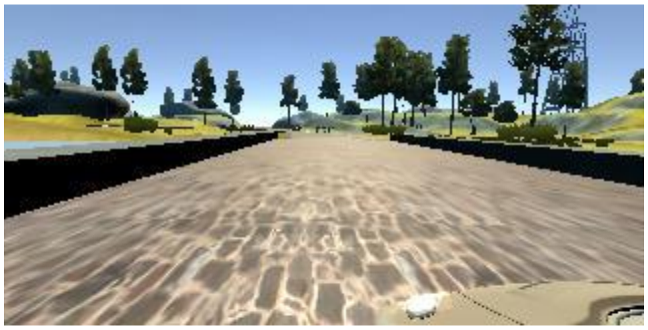


I took the advantage of the left and right camera to teach the car how to handle sharp turns. By adding a correction factor to the steering angles for the left camera images and substracting a correction factor to the steering angles for the rigth camera images, the car can learn to turn more at the sharp turns.
The three images shown below are the images produced by the center, left and right cameras:

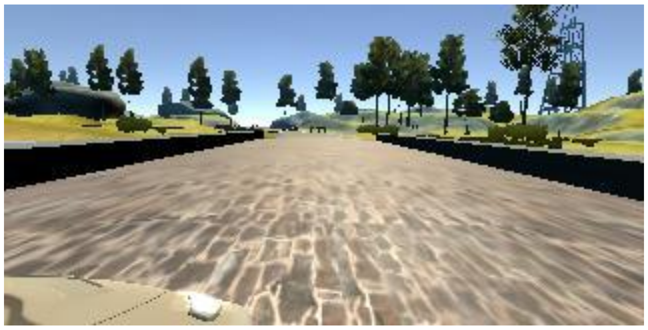
Center view:



Left view:



Right view:



I have also tried data augmentation (see model.py line 45-57). However, after several experiments, the car performed very poor after applying data augmentation. Therefore, I decided not to use data augmentation for training the model.

After the collection process, I had I collected 35,265 training images in total which was proven to be sufficient for training the neural network. For data preprocessing, I added a lambda layer to normalize the data set. In addition, I cropped each image by removing the top 70 pixels and the bottom 25 pixels. Then I shuffled the data set and split the data set into training data set and validation data set. 80% of the data was put into training data set and the rest was put into validation data set. After the model was trained using the training data, the model was evaluated using validation data to see if the model was underfitting or overfitting. After several trainings, the ideal number of training epochs for my model was 10. The learning rate was not tuned since an adam optimizer was applied.