Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Build, a convolution neural network in Keras 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

Rubric Points

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

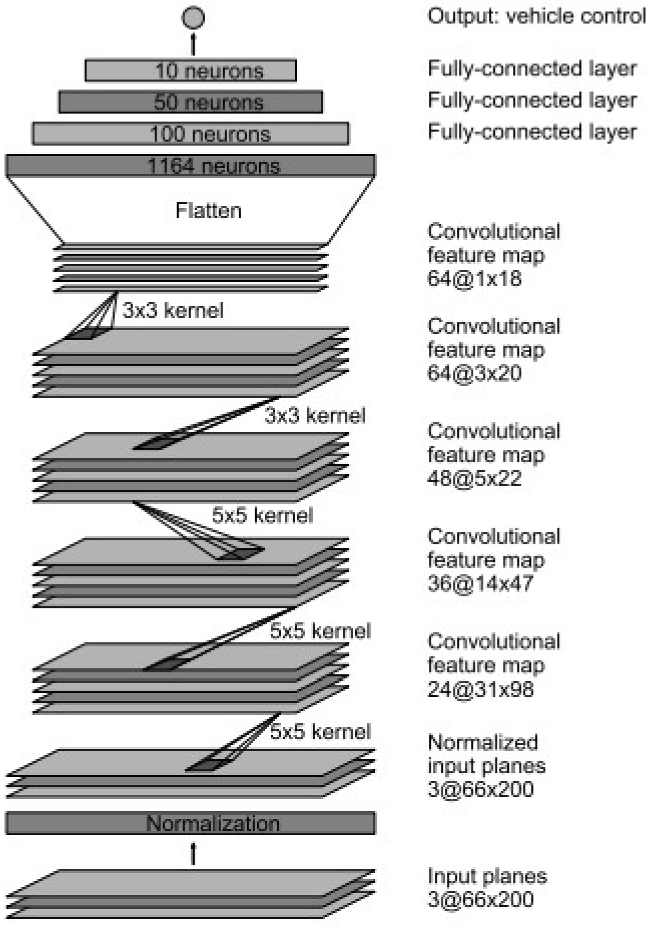
model.py containing the script to create and train the model (simple code mostly derived from what is presented in class)
drive.py for driving the car in autonomous mode (modified the speed from 9 to 15- only change to the file provided on Github, more on this later).
model.h5 containing a trained convolution neural network
writeup_report.pdf summarizing the results

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. Submitted code is readable with appropriate comments and the files are functionable

Model Architecture and Training Strategy

The model consists of a convolution neural network, as discussed in the Nvidia paper illustrated below (https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf). I have used the same model with ELU activation and drop-outs to avoid over- fitting.

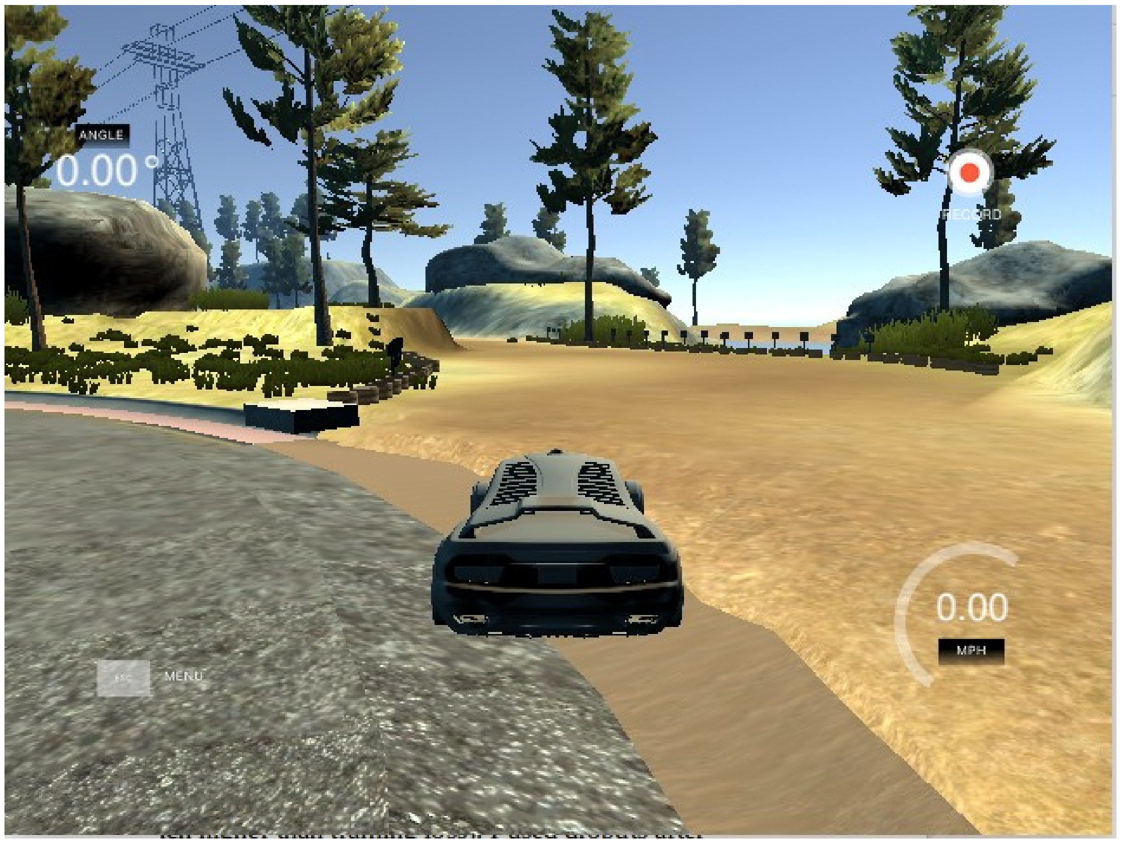
The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. However, the input image was not split into YUV plane; instead the regular RGB format is passed to the network. The model includes ELU layers to introduce nonlinearity, and the data is normalized, cropped in the model using a Keras framework.



2. Attempts to reduce overfitting in the models

I worked with the training set (provided by udacity, only center images). This resulted in overfitting with the above architecture (validation loss much higher than training loss). I used droputs after activation functions to mitigate overfitting. In addition the training data set was enhanced using images from all three cameras and flipping left to right of each image.

The model was trained and validated on different data sets and shuffling to ensure that the model was not overfitting. The model used an adam optimizer, so the learning rate was not tuned manually. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. The key item is that even with stable and matched training / validation losses. Vehicle could not stay on track particularly here in autonomous mode (location showed in training mode for convenience. It used to go straight.

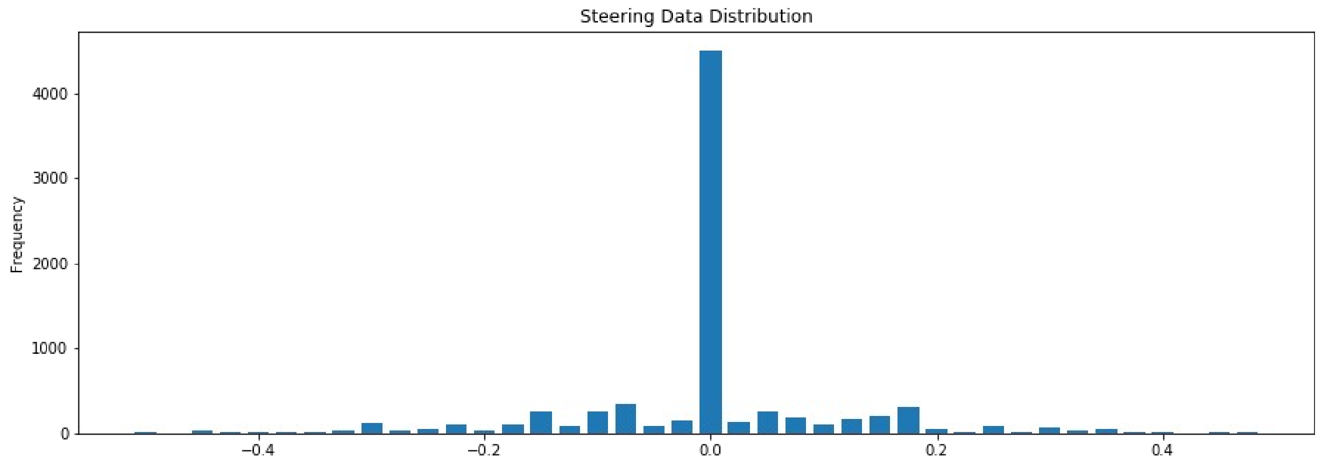


[illegible]

An example of what was happening in the first attempt of the model

4. **Appropriate training data**

Udacity training data was chosen. Histogram shows that the data is heavily skewed towards steering angle 0 (using only images of center camera). The trained model also has preference to zero steering angle. This could be one of the main reasons in missing the track above. I wanted to make the track work without adding data at these locations to the udacity data.

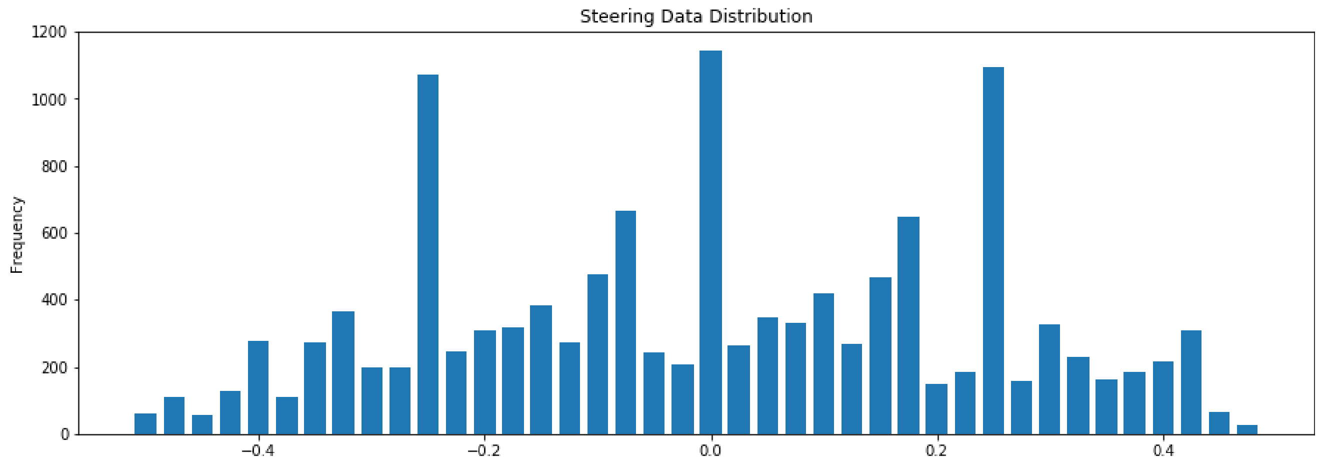


I randomly dropped 80% of the data with steering angle zero so that the data is not heavily skewed. I have spent considerable time to implement this in Python. Finally, I used pandas with help from online for this step.

Later I followed the course lecture in using the rest of the images (center, left and right) with a steering correction of +/- 0.25 applied to left and right camera images.

All the image data is immediately converted to RGB format after reading the image with cv2

The resultant data distribution as follows:



Overall numberf of samples (images after screening)
`len(measurements) = (13641,)`

The above data was randomly shuffled and 20% of the above data is assigned to validation set and the rest to training data.

Model Architecture and Training Strategy

####1. Solution Design Approach

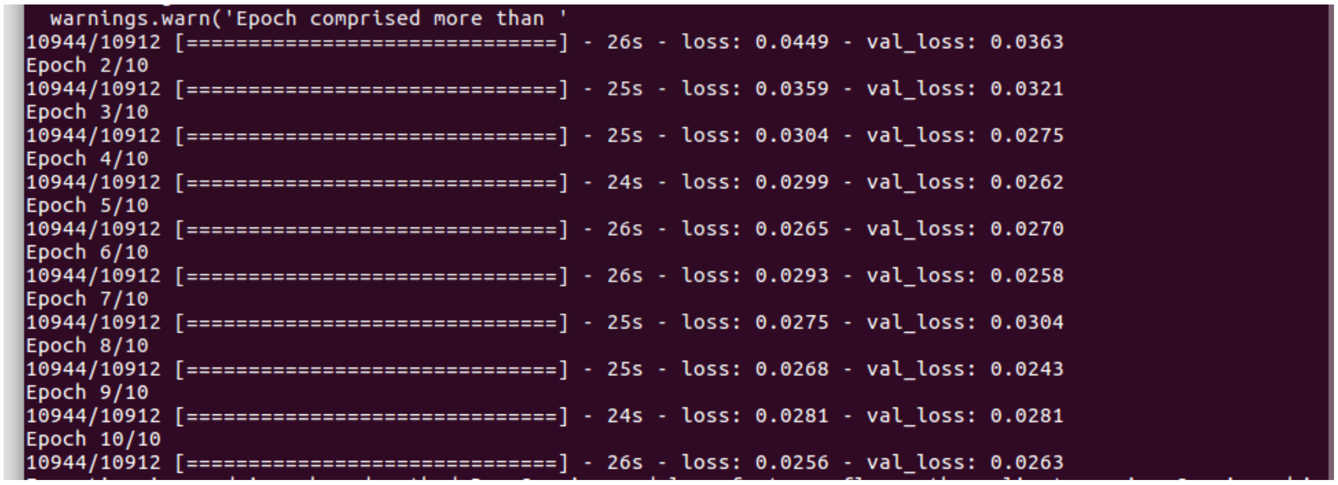
The model used here is the same by Nvidia. Keras framework was used to normalize and crop the image which was then sent through 5 convolution layers and 4 fully-connected layers to get steering angle estimate. MSE was used as a loss function, and adam optimizer was used to train the model.

A Python generator is used to generate data for training and validation rather than storing the entire data in memory. The model.py code is clearly organized and comments are included .

Model.fit_generator is used to train the model. Training loss and Validation loss are calculated; validation loss was higher than training loss, which confirms over-fitting . Drop-outs were introduced gradually post activation to mitigate over-fitting. A single hyper-parameter “r” is defined as drop-out rate, and the same rate is applied to all drop-outs post activation.

The key item is that I have been tracking 2 items with each additional drop-out:
a) comparison of the training and validation loss vs each epoch
b) testing the model.h5 in autonomous mode.

I added dropout with a fixed rate for all layers post activation except the last FC layer.
The rate of 0.12 is chosen iteratively to minimize the delta between training and validation loss.
10 epochs were run for this, and a snap shot of the losses are included.



```
  warnings.warn('Epoch comprised more than '
10944/10912 [==============================] - 26s - loss: 0.0449 - val_loss: 0.0363
Epoch 2/10
10944/10912 [==============================] - 25s - loss: 0.0359 - val_loss: 0.0321
Epoch 3/10
10944/10912 [==============================] - 25s - loss: 0.0304 - val_loss: 0.0275
Epoch 4/10
10944/10912 [==============================] - 24s - loss: 0.0299 - val_loss: 0.0262
Epoch 5/10
10944/10912 [==============================] - 26s - loss: 0.0265 - val_loss: 0.0270
Epoch 6/10
10944/10912 [==============================] - 26s - loss: 0.0293 - val_loss: 0.0258
Epoch 7/10
10944/10912 [==============================] - 25s - loss: 0.0275 - val_loss: 0.0304
Epoch 8/10
10944/10912 [==============================] - 25s - loss: 0.0268 - val_loss: 0.0243
Epoch 9/10
10944/10912 [==============================] - 24s - loss: 0.0281 - val_loss: 0.0281
Epoch 10/10
10944/10912 [==============================] - 26s - loss: 0.0256 - val_loss: 0.0263
```

This model successfully finished the track in autonomous mode.
Drive.py was not modified as all the image processing (normalization and cropping) was done in the model. Only parameter that I played is with set speed.

It was the only parameter I changed. I set different speeds to stress where the car goes off the track completely. Set speed up to 20 drive without any issues. Set speed of 25 also finishes the track, but it touches the track boundaries at certain locations.
For this project, I generated the video with set speed of 15 and 60 fps.

Really loved this project, I know that there is scope for further improvement to reduce wobbly. The key is to have lot of training data (with right-preprocessing and augmentation). Over-fit to minimize training loss and then variable drop-our rates to reduce validation loss.