Behavioral Cloning

Behavioral Cloning Project

Goal

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 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

Report

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
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- report.md summarizing the results (which you are reading right now.)

2. Submission includes functional code

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
```

3. Submission code is usable and readable

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 and Training Strategy

1. An appropriate model architecture has been employed

For this assignment, I had initially started off with a simple approach to see if the code was working. While there were several approaches that could have been undertaken, I planned to use the NVIDIA architecture mentioned in the lectures. I also stumbled upon a Comma.AI team's model that seemed to be quite simple on the outset. Thus I tried it out first to see how different it is from the NVIDIA model.

Reference:
NVIDIA model: model.py lines 46-80
Comma.AI model: model.py lines 21-41

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting. In the NVIDIA model, L2 regularization was also used to tune the weights.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 10-16). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

Both the models used an adam optimizer, so the learning rate was not tuned manually (model.py line 25).

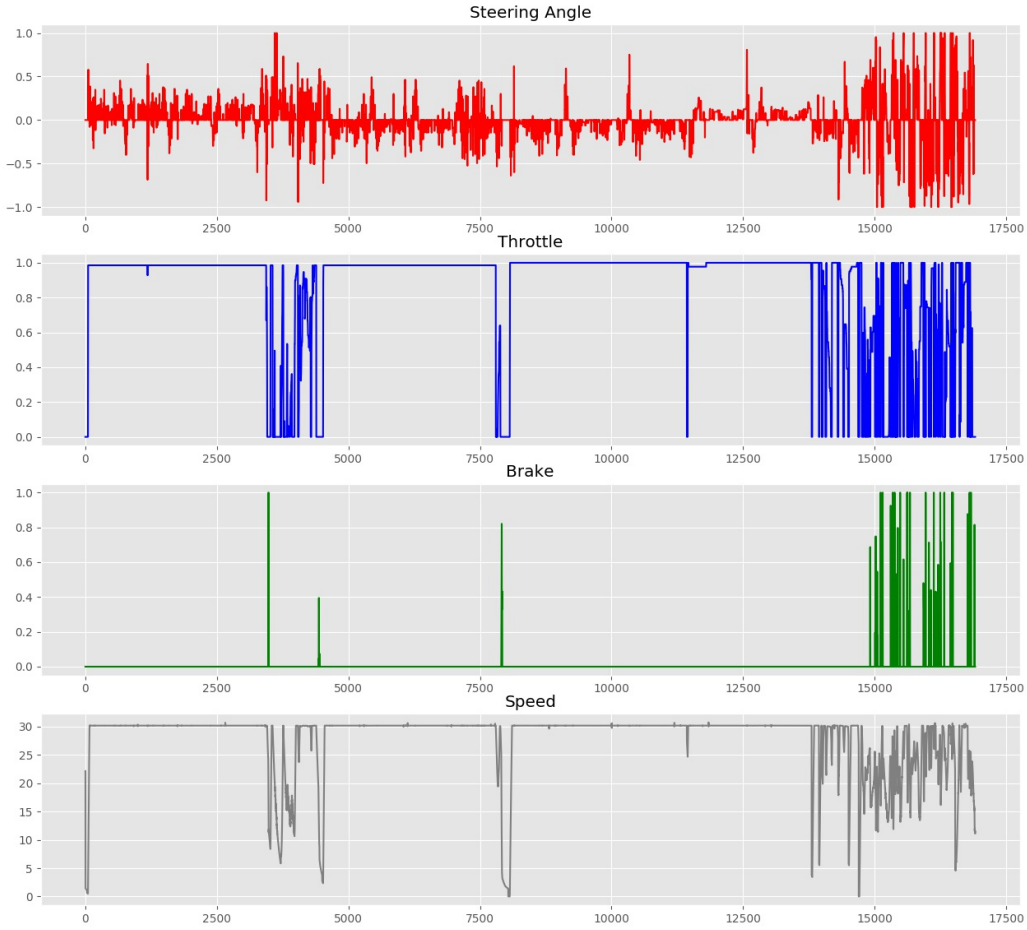
4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I initially started off with center lane driving. I tried both keyboard and mouse to gather the data but it was not that easy. So, I hooked up my old Xbox 360 controller and configured the simulator to pick up the controller and got to creating more training data.

In total I performed the following to gather the training data:

- Track 1 center lane in the right way - 2 laps
- Track 1 center lane in the opposite direction - 2 laps
- Track 1 recovery from left lane - 2 times
- Track 1 recovery from the dirt area on left and on right - 2 times
- Bridge edge to center - Once
- Track 1 just off bridge the dust area once again - once
- Track 2 One full lap - tried to stay in the right side lane as much as I could.

Below is a graphical representation on the training data that I had. Funnily enough, the throttle, brake and speed are insignificant to this training model as we are only predicting the steering angle.



As you can see, there seems to be a decent spread on the images that can be used.

For details about how I created more training data, see the section Creation of the Training Set & Training Process.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to look at a good predefined model as inferred from the lectures. This is purely my inference. Having many good off the shelf models, I wanted to concentrate more on learning how to gather data and select a good model for my problem rather than tune each and every layer of my neural network

Before looking at the various approaches like LeNet, NVIDIA and the CommaAI that I later found, I started off with a simple convolution neural network with a 5x5 filter and 2,2 slide. This helped me understand if my model was even generating an output for me to use. Upon testing this model (for educational purposes), the vehicle expectedly jumped into the river.

Adding a couple of layers and tweaking the neural network led to the same set of going in circles on the dirt area or driving straight underwater. Then it stuck me that the `drive.py` code was using RGB images and I was training on BGR images. A single change in `drive.py` code ensured that I was staying on the road. Not for long tough as right after the first curve (while staying on the edge) I straight off got stuck at entrance of the bridge in Track 1.

Looking at the MSE (Mean squared error) on both my training and validation data (Training = 0.06 and Validation = 0.16), it was clear that there was overfitting going on. Till this point, I was using the Udacity provided sample images. I then created a helper function to pull in all the data (Udacity samples and my training data) and tested with this and found that the overfitting of the data reduced to a certain extent.

This was not enough, so I stareted to perform some data augmentation (basic flipping the images) and adding the left and right cameras into the equation and ensuring that the steering measurements were being taken into proper consideration.

At this point, I got the car to move to a certain distance before it looses its mind. I had added some simple Conv2D layers and Fully connected layers with a single drop out layer right after flattening the Conv2D layer output. This was the extent of me trying to fudge around with the neural network.

Then I went ahead and started playing around with the CommaAI model. The results were pretty good. There was some visible improvements to the way in which the car was not swaying a lot and was able to tackle most of the corners without going into the dirt. At this point, I implemented some Keras enabled cropping to the input layer to remove unneeded data. This helped the model quite a bit and I was able to complete the track for more than 2 laps successfully.

To satisfy my curiosity I went ahead and searched for the NVIDIA paper mentioned in the lectures and tried to implement it. At the first run, without reading the paper properly, the car promptly decided to stay straight in line and run into the ocean after attempting the first corner. Then reading the paper helped me understand that they had trained the model in the YUV color space. Failing

to update the `drive.py` file yielded me quite similar results.

Finally I updated the `drive.py` file to work for either model with the help of a flag.

At the end of the process, the vehicle is able to drive autonomously around the first track without leaving the road.

As for the second track, the CommaAI model was not quite up to the task and kept failing immediately (mostly due to the lack of Track 2 training data). Surprisingly NVIDIA model (limiting the top speed to 12 MPH) went quite far with the same amount of training data that was provided to the CommaAI model, but as expected, did not complete and fell off the cliff after completing nearly 60-75% of the track.

My guess is I need to provide a little bit more training for the CommaAI model to pick up track 2 which has quite a few turns and elevations.

2. Final Model Architecture

As mentioned in the above section, I used both the NVIDIA and CommaAI models to compare both the methodologies. The architectures are as follows:

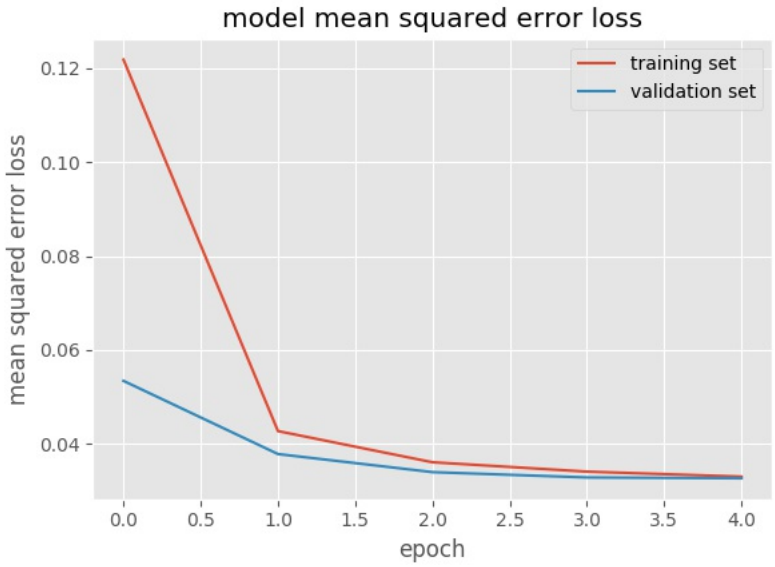
NVIDIA Architecture

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 160, 320, 3)   0           lambda_input_1[0][0]
____________________________________________________________________________________________________
cropping2d_1 (Cropping2D)        (None, 65, 240, 3)    0           lambda_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 31, 118, 24)   1824        cropping2d_1[0][0]
____________________________________________________________________________________________________
elu_1 (ELU)                      (None, 31, 118, 24)   0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 14, 57, 36)    21636       elu_1[0][0]
____________________________________________________________________________________________________
elu_2 (ELU)                      (None, 14, 57, 36)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 5, 27, 48)     43248       elu_2[0][0]
____________________________________________________________________________________________________
elu_3 (ELU)                      (None, 5, 27, 48)     0           convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 3, 25, 64)     27712       elu_3[0][0]
____________________________________________________________________________________________________
elu_4 (ELU)                      (None, 3, 25, 64)     0           convolution2d_4[0][0]
____________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 1, 23, 64)     36928       elu_4[0][0]
____________________________________________________________________________________________________
elu_5 (ELU)                      (None, 1, 23, 64)     0           convolution2d_5[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 1472)          0           elu_5[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           147300      flatten_1[0][0]
____________________________________________________________________________________________________
elu_6 (ELU)                      (None, 100)           0           dense_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        elu_6[0][0]
____________________________________________________________________________________________________
elu_7 (ELU)                      (None, 50)            0           dense_2[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            510         elu_7[0][0]
____________________________________________________________________________________________________
elu_8 (ELU)                      (None, 10)            0           dense_3[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             11          elu_8[0][0]
====================================================================================================
Total params: 284,219
Trainable params: 284,219
Non-trainable params: 0
____________________________________________________________________________________________________
```

CommaAI architecture

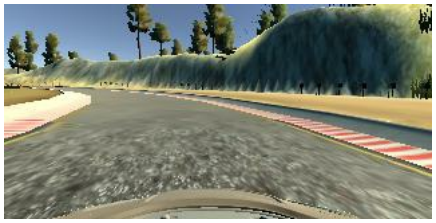
```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 160, 320, 3)   0           lambda_input_1[0][0]
____________________________________________________________________________________________________
cropping2d_1 (Cropping2D)        (None, 65, 240, 3)    0           lambda_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 17, 60, 16)    3088        cropping2d_1[0][0]
____________________________________________________________________________________________________
elu_1 (ELU)                      (None, 17, 60, 16)    0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 9, 30, 32)     12832       elu_1[0][0]
____________________________________________________________________________________________________
elu_2 (ELU)                      (None, 9, 30, 32)     0           convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 5, 15, 64)     51264       elu_2[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 4800)          0           convolution2d_3[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 4800)          0           flatten_1[0][0]
____________________________________________________________________________________________________
elu_3 (ELU)                      (None, 4800)          0           dropout_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 512)           2458112     elu_3[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 512)           0           dense_1[0][0]
____________________________________________________________________________________________________
elu_4 (ELU)                      (None, 512)           0           dropout_2[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 1)             513         elu_4[0][0]
====================================================================================================
Total params: 2,525,809
Trainable params: 2,525,809
Non-trainable params: 0
____________________________________________________________________________________________________
```

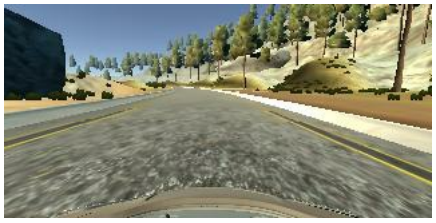
NVIDIA model loss Visualization

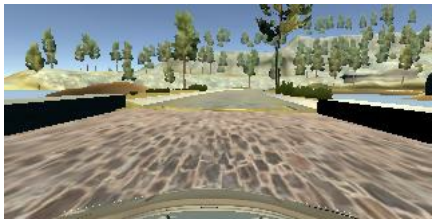


3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded a total of four laps on track one using center lane driving. Both in the correct direction and in the opposite directions. Here are a few example images of center lane driving:







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 behave in such scenarios. This was in particularly useful after I tried the model for a few different times and each and every time it went off the track and didn't know how to recover itself. Some of the sample images for the recovery:

Example of recovering from the left side, going off the track:

Another example for recovering from the right side:

I performed a simple centered line driving on Track 2 to gather some more information from that track as it involved quite a bit of extreme turns which the Track 1 lacked severely. I performed only 1 single run as I wanted to see how the model would perform with less training data in a relatively newer environment.

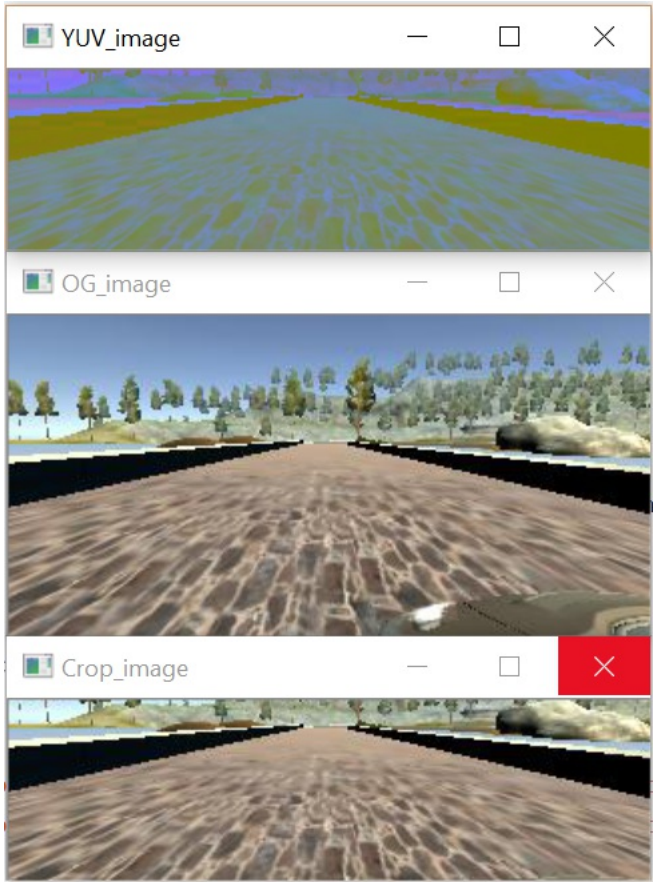
In order to increase my data set that I can provide the model for training, I performed some simple augmentation by flipping the images in question. This would literally look like the mirror of each other, but would increase the training data by 2 fold.

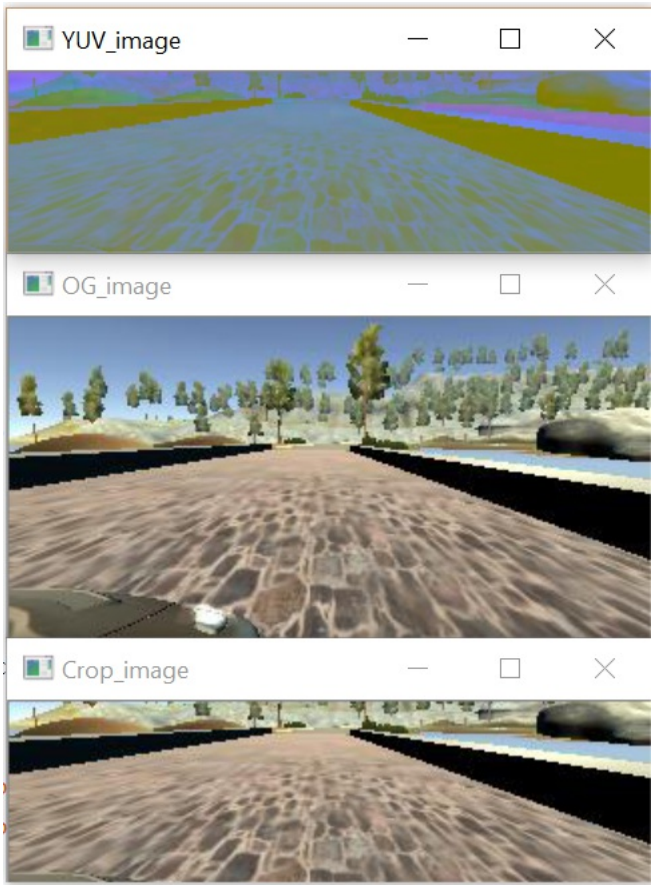
I did not perform any other sort of brightness or warping augmentation for this project as its behavior cloning on how you drive. The more behavior you feed, theoretically should improve the model/system rather than just dumping it with fake data.

At this point, I had my model fed the right color space images and the required cropping using Keras instead of OpenCV as it is better to use the GPU rather than the CPU for these processes.

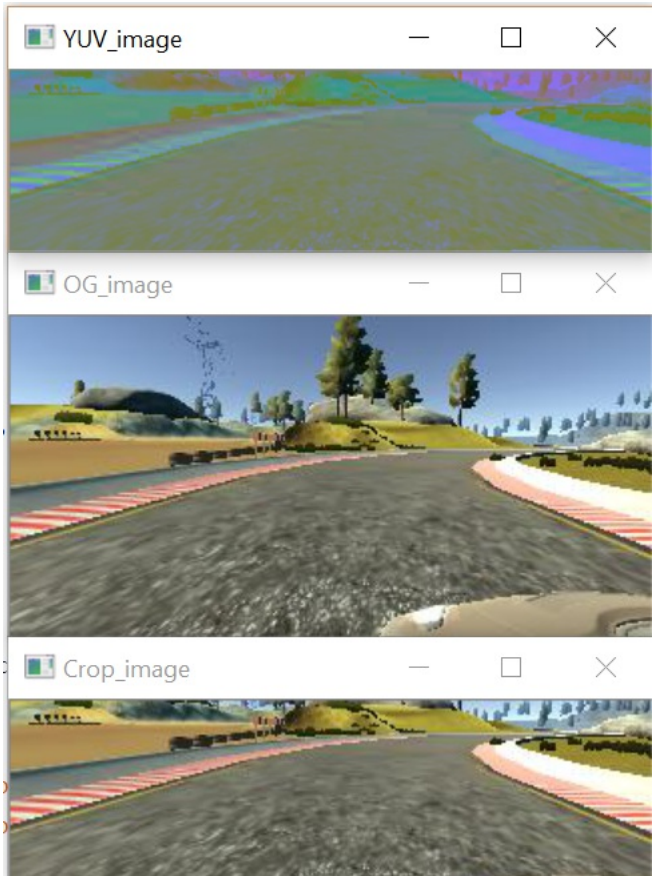
Some sample random images: They are the Original images, cropped, and in YUV color space as ingested by the NVIDIA model.


YUV_image
OG_image
Crop_image


YUV_image
OG_image
Crop_image


YUV_image
OG_image
Crop_image


YUV_image
OG_image
Crop_image



At this point, with all 3 camera angles and simple flipping of the images, I ended up with nearly 105K images.

I finally randomly shuffled the data set and put 20% of the data into a validation set and fed this into the Generator for the Model to pull them up as and when needed. This meant nearly 81K training images and 20K images to validate.

As for the number of Epochs, I decided to trust the instructors in the statement that 5 Epochs are quite enough for this model and it turned out so after experimenting a little with the Epoch numbers. I used an adam optimizer so that manually training the learning rate wasn't necessary.

Simulation

Is the car able to navigate correctly on test data?

As seen from the videos present in the `videos` folder, both the models were able to successfully lap the Track 1 atleast once.

As for Track 2, CommaAI failed at the initial try itself. NVIDIA model was able to complete 60-75% of it when the speed was capped off at 12 MPH.