Behavioral Cloning

Writeup Template

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
- Behavioral Cloning WriteUptemplate V1.pdf summarizing the results
- video.mp4 - A video recording of the car driving autonomously at least one lap around the track

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.

Model Architecture and Training Strategy

1. **An appropriate model architecture has been employed**

I spent a lot of time exploring different model architectures. I looked at the Nvidia model but noticed it had long training time. I settled for a neural network with 3 convolutional layers and 3 fully connected layers (model.py lines 206 to 219).

I added dropout on 2 out of 3 dense layers to prevent overfitting. The model was trained using Adam optimizer with a learning rate = 1e-04 and mean squared error as a loss function. I used 20% of the training data for validation and the model performed well after training for 2 epochs**.**

The the data is normalized in the model using a Keras lambda layer (model.py code line 203). I used the same code as the one used in the lectures. I also added a Cropping2D layer (model.py code line 204).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 214, 216).

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model was trained using Adam optimizer with a learning rate = 1e-04 and mean squared error as a loss function. (model.py line 219).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used the training data provided by Udacity and it was good quality data. The data has 8036 images from center camera, 8036 from left camera and 8036 from right camera.

Model Architecture and Training Strategy

1. Solution Design Approach

I settled for a neural network with 3 convolutional layers and 3 fully connected layers (model.py lines 206 to 219). I tried different model architectures, however, this was the one that took the least processing time. I added dropout on 2 out of 3 dense layers to prevent overfitting.

The final step was to run the simulator to see how well the car was driving around track one. Initially I used only the center images. During the first run, my car did not even get past the first turn and fell into the water. I realized I had to improve the model or add more image pre-processing and also use more images.

I upsampled left and right turns by doubling the sample to improve the ability to predict turns. I also downsampled near zero steering by discarding 10% of the data. (model.py rows 64-92).

Then I also applied a correction for left and right steering angles and randomly selected the center, left or right image (model.py rows 141-162). I generated another model. This model also did not get the car to complete the track, however, the performance was much better.

After that, as suggested in the lectures, I added code to flip the images to help with the left turn bias. I got best performance by randomly flipping 80% of the images (model.py rows 165 to 170).

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

2. Final Model Architecture

The final model architecture (model.py lines 206 to 219) consisted of a neural network with 3 convolutional layers and 3 fully connected layers.

I added dropout on 2 out of 3 dense layers to prevent overfitting. The model was trained using Adam optimizer with a learning rate = 1e-04 and mean squared error as a loss function. I used 20% of the training data for validation and the model performed well after training for 2 epochs**.**

The data is normalized in the model using a Keras lambda layer (model.py code line 203). I used the same code as the one used in the lectures. I also added a Cropping2D layer (model.py code line 204).

Model Summary:

```
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 160, 320, 3)       0
_________________________________________________________________
cropping2d_1 (Cropping2D)    (None, 65, 320, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 63, 318, 16)       448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 159, 16)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 29, 157, 32)       4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 78, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 76, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 38, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 14592)             0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               7296500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               50100
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 20)                2020
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 21
=================================================================
Total params: 7,372,225.0
Trainable params: 7,372,225.0
Non-trainable params: 0.0
_________________________________________________________________
```

3. Creation of the Training Set & Training Process

The training data I used was the data provided by Udacity. The data was pretty good quality.

Here are some characteristics of the training data:

➔No. of samples—8036 images from center camera, 8036 from left camera and 8036 from right camera.

➔The data had steering, speed, break and throttle for each of the 8036 images.

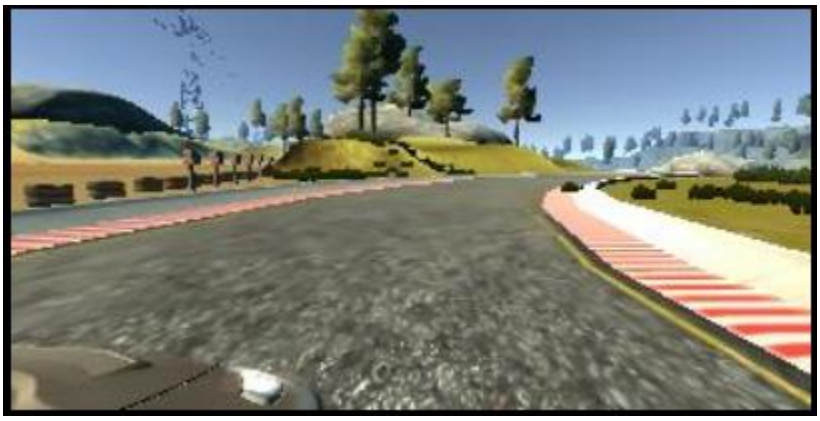
➔All images are 320x160 pixels

Here are some images from the data:

Center Image:

Left Image:

Right Image:



This dataset of 8036 images was a limited sample so I decided to perform a lot of pre-processing on the dataset.

Here is some of the pre-processing that I did –

➔Upsampled left and right turns (model.py lines 64 to 80): The dataset contained only a little bit of non-zero steering data, so I upsampled left and right turns by doubling the sample to improve the ability to predict turns.

➔Downsampled near zero steering (model.py lines 83 to 88): I downsampled near zero steering by discarding 10% of the data.

➔Included left and right images (model.py lines 112 to 131): To enable recovery I used data from all 3 cameras. As suggested in the lectures I applied a steering correction factor of 0.20 (model.py lines 135-138). Also I randomly chose only one of the images (center, left or right, model.py lines 141 to 162).

➔Flipping images (model.py lines 165 to 170): As suggested in the lectures, images were randomly flipped and the steering was reversed. I got best performance by randomly flipping 80% of the images.

At the end of pre-processing, I had 15,344 images and angles.

Also, within the Keras model, the data was normalized using a Keras lambda layer (model.py code line 203). I used the same code as the one used in the lectures. I also added a Cropping2D layer (model.py code line 204).

Final thoughts:

This was a very challenging and time consuming project. I had initially decided to use generators, however, I found that using generators was making the code slower to process. So, I decided not to use generators. I also found that generating recovery data in the simulator was very challenging using only a mouse and the data generated did not help improve the model.

I used the beta-simulator for testing.

I am grateful to Udacity to providing the sample data which was excellent.

Overall, it was a wonderful experience, I learned a lot and I am happy with the outcome.

Useful Blogs and websites used:

https://discussions.udacity.com/c/nd013-deep-learning/nd013-project-behavioral-cloning

https://carnd.slack.com/messages/C2HQV18F2/search/driving%20log%20format/

https://medium.com/@mohankarthik/cloning-a-car-to-mimic-human-driving-5c2f7e8d8aff#.6rraficai

https://medium.com/@priya.dwivedi/teaching-a-car-to-drive-itself-8c612604fdc5#.qq0hbt5zi

https://medium.com/self-driving-cars/6-different-end-to-end-neural-networks-f307fa2904a5#.7zmuqzjpr

https://carnd-forums.udacity.com/cq/viewquestion.action?id=26214464&questionTitle=behavioral-cloning-cheatsheet

https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9

https://github.com/

http://stackoverflow.com/