Behavioral Cloning

Udacity CarND

1. Overview

Behavioral Cloning Project

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

2. Project Introduction

This project is using **Udacity Simulator** for driver behavioral cloning. After the data generator from simulator, the convolutional neural network will be able to learn from those data. Finally, those data can teach the car to drive itself automatically in the simulator.

3. Project Pipeline

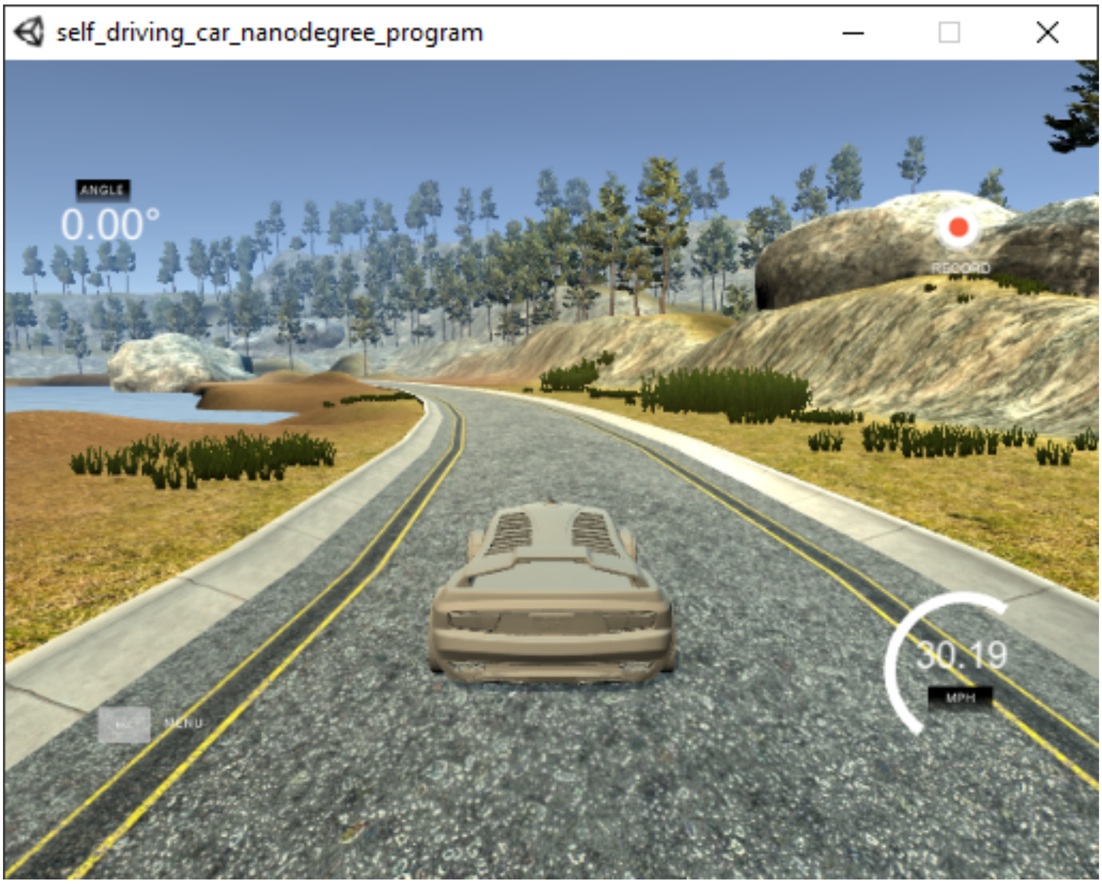
The project basically follows the following steps:

1. Drive the car in training mode in simulator, try to keep the car in the middle of the road
2. `model.py` reads the data through the convolutional neural network and saves the

parameters in `model.h5`
3. Using generated `h5` file to run the simulator in autonomous mode
4. Generate the video during the autonomous driving in simulator

3.1 Data generator

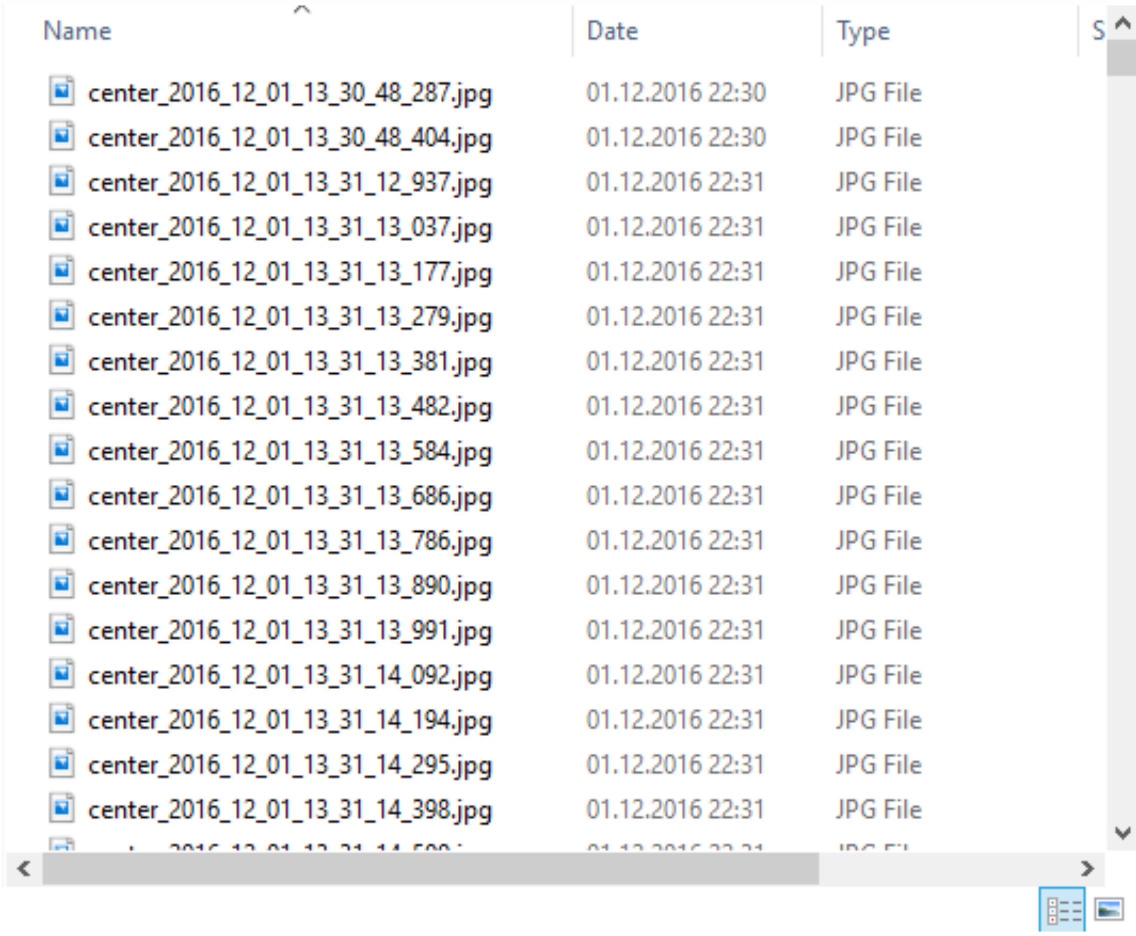
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:



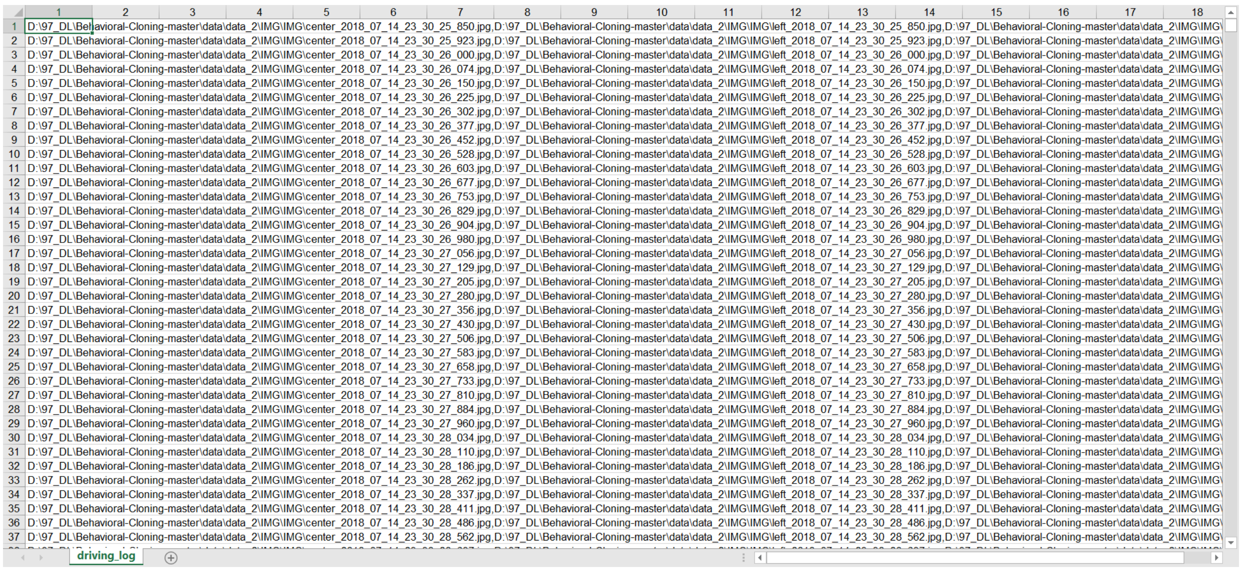
I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to go back to the middle of the road when the car began to off center.

Then I repeated this process on track two in order to get more data points.

The `model.py` in this project is using the generated data from **Udacity**.



The new data is also generated, but they are not tested.



3.2 Model Architecture

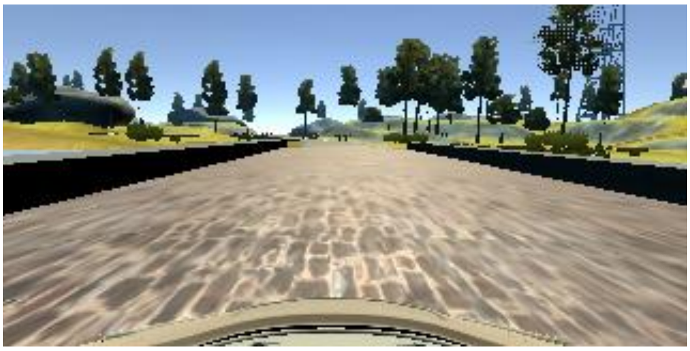
3.2.1 Image preprocess

The first step is to read the `.csv` file and the related images. The images and the labels (the steering angle) are saved to the variables `X_train` and `y_train`.

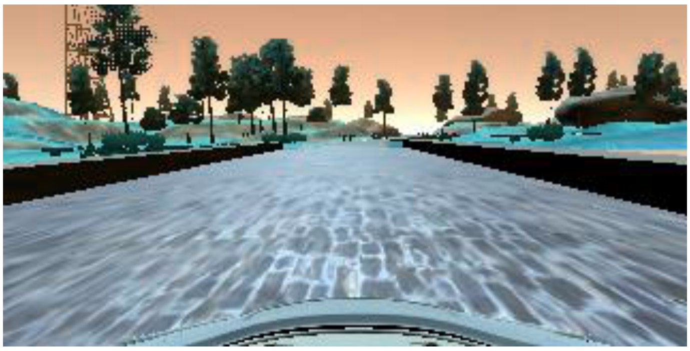
The code, `X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=0.2, random_state=0)`, splits the `X_train` and the `y_train` to the trainning examples and the validation examples.

To augment the data sat, I flipped images and angles.

Original image with original angles:



Flipped image with (original angles $\times(-1)$*)*:



After the collection process and `train_test_split(...)`, I had *38572* training examples and *9644* validation examples instead of *19286* training examples and *4822* validation examples. I randomly shuffled the data set and put 20% of the data into a validation set.

3.2.2 Neural network model

I used 5 convolution layers with batch normalization and activation ELU, 1 Flatten layer and 3 dense layer with activation ELU and using dropout to not overfit the network. *line 107 - line 152* in `model.py`

The input layer is image with shape $160 \times 320 \times 3$ and the output layer size is 1×1. In the input layer, the image is resized to $60 \times 120 \times 3$.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
cropping2d_1 (Cropping2D)    (None, 60, 320, 3)        0         
_________________________________________________________________
lambda_1 (Lambda)            (None, 60, 120, 3)        0         
_________________________________________________________________
lambda_2 (Lambda)            (None, 60, 120, 3)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 60, 120, 3)        12        
_________________________________________________________________
elu_1 (ELU)                  (None, 60, 120, 3)        0         
_________________________________________________________________
```

```
batch_normalization_1 (Batch (None, 60, 120, 3)        12
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 60, 16)        1216
_________________________________________________________________
elu_2 (ELU)                  (None, 30, 60, 16)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 30, 60, 16)        64
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 30, 32)        12832
_________________________________________________________________
elu_3 (ELU)                  (None, 15, 30, 32)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 15, 30, 32)        128
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 15, 64)         18496
_________________________________________________________________
elu_4 (ELU)                  (None, 8, 15, 64)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 15, 64)         256
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 8, 128)         73856
_________________________________________________________________
elu_5 (ELU)                  (None, 4, 8, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 4, 8, 128)         512
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
elu_6 (ELU)                  (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               2097664
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
elu_7 (ELU)                  (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               51300
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
elu_8 (ELU)                  (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
_________________________________________________________________
dropout_3 (Dropout)          (None, 10)                0
_________________________________________________________________
elu_9 (ELU)                  (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
```

```
=================================================================
Total params: 2,257,369
Trainable params: 2,256,883
Non-trainable params: 486
```

3.2.3 Attempts to reduce overfitting in the model

The following techniques can avoid the overfitting:

- Normalization in input layer (*line 64 - line 68 in* `model.py`)
- Batch Normalization for each layer
- Dropout on the dense layers (fully connection layers)

The model was trained and validated on different data sets to ensure that the model was not overfitting (*line 155*). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3.2.4 Model parameter tuning

The model used an adam optimizer. The adam parameter `lr` chooses 10^{-5} so the learning rate was not tuned manually (`model.py` *line 149*).

3.2.5 Model save

After training data on GPU, the model is saved to `model.h5`. `drive.py` will call `model.h5` when I use the simulator in *autonomous mode*. Based on the parameter I tried, the car in *autonomous mode* is able to drive by itself and tries to keep itself in the middle of the road.

4. Let's run the code

After building the model in `model.py`, I run it on a GPU computer with **batch_size** 32, **epochs** 50. Some of the epochs show below:

```
Train on 38572 samples, validate on 9644 samples
Epoch 1/100

32/38572 [..............................] - ETA: 1:09:32 - loss: 7.2933 - acc: 0.0000e+00
64/38572 [..............................] - ETA: 35:48 - loss: 7.3409 - acc: 0.0156
96/38572 [..............................] - ETA: 24:32 - loss: 7.1540 - acc: 0.0208
128/38572 [..............................] - ETA: 18:54 - loss: 9.0754 - acc: 0.0156
160/38572 [..............................] - ETA: 15:31 - loss: 8.2795 - acc: 0.0312
...
Epoch 50/50
```

```
32/38572 [..............................] - ETA: 1:06 - loss: 0.0364 - acc: 0.1875
64/38572 [..............................] - ETA: 1:08 - loss: 0.0379 - acc: 0.2656
96/38572 [..............................] - ETA: 1:08 - loss: 0.0347 - acc: 0.2292
128/38572 [..............................] - ETA: 1:08 - loss: 0.0328 - acc: 0.2188
...
38432/38572 [============================>.] - ETA: 0s - loss: 0.0312 - acc: 0.1818
38464/38572 [============================>.] - ETA: 0s - loss: 0.0312 - acc: 0.1819
38496/38572 [============================>.] - ETA: 0s - loss: 0.0312 - acc: 0.1819
38528/38572 [============================>.] - ETA: 0s - loss: 0.0312 - acc: 0.1819
38560/38572 [============================>.] - ETA: 0s - loss: 0.0312 - acc: 0.1819
38572/38572 [==============================] - 79s 2ms/step - loss: 0.0312 - acc: 0.1818 - val_loss: 0.0185 - val_acc: 0.1754
```

`checkpointer` and `model.save(...)` can save the model and trained parameters in **model.h5**.

Open the simulator and choose the *autonomous mode*, run the script in the *Terminal* or the *cmd*:

```
python drive.py model.h5 run1
```

The car is able to drive by itself.

To generate the video based on the images in `run1` folder, run the script:

```
python video.py run1
```

```
Internet:CarND-Behavioral-Cloning-P3-master Haiqi$ python3 video.py run1
Creating video run1, FPS=60
[MoviePy] >>>> Building video run1.mp4
[MoviePy] Writing video run1.mp4
100%|██████████████████████████████| 8404/8404 [00:41<00:00, 204.32it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: run1.mp4
```

Here is the video of the autonomous driving in the simulator: [link to my video result]