**CarND-P3-Report**

**Data Generation**

The vehicle was drove through Track 1 twice, one time clockwise and the second time counter-clockwise. During driving, I tried to weave back and forth between the middle of the road and the shoulder to mimic the real-world scenario.

To generate more data, as suggested by the lecture, correction (0.2) was applied to the center steering angle to utilize the images from left and right cameras.

**Data Preprocessing**

The images were normalized to (-1, 1) first. After that the top 50 and the bottom 20 rows of pixels were cropped off because of the useless objects.

**Model Architecture and Training**

Python generator is used to generate the training and validation batches instead of importing the whole data in memory.

The basic architecture of the model was from the NVIDIA autonomous driving paper. It includes five convolution layers and three fully connected layers. The architecture was realized using Keras.

Multiple experiments were conducted to decide the size of each layer. It is observed that it is mainly the data quality not the model architecture that impacts the model results. Finally the size of each layer is as follows:

|  |  |  |
| --- | --- | --- |
| Layer | Filter size | Layer size |
| Input layer |  | (90, 320, 3) |
| Convolution layer 1 | (24, 5, 5) | (32, 156, 24) |
| Convolution layer 2 | (36, 5, 5) | (14, 76, 36) |
| Convolution layer 3 | (48, 5, 5) | (5, 36, 48) |
| Convolution layer 4 | (64, 3, 3) | (3, 34, 64) |
| Convolution layer 5 | (64, 3, 3) | (1, 32, 64) |
| Flatten layer |  | 2048 |
| Fully connected layer 1 |  | 100 |
| Fully connected layer 2 |  | 50 |
| Fully connected layer 3 |  | 10 |
| Output |  | 1 |

The loss objective function is “mse”, and the optimizer is “adam”. The number of epochs is set as “3”.

**Results**

The program was run on my local machine. It took about more than 3 minutes for each epoch (In total around 10 minutes). Finally the best validation MSE loss is 0.0074.

The performance of the model under autonomous model is shown in the run video.

**Problems Met**

**Tensorflow out of memory**

**Failed Explorations**

I tried to consider the images as a time series to generate the training dataset. Simply speaking, the images at time step and are concatenated together as a new image, which then corresponds to the steering angle at . The other process was kept the same.

I guess it will make sense because if there was a curve on the road, the steering angle should be impacted for a few time steps. However, the local computer cannot afford this heavy computation (It crashed during the training). The validation loss was not obviously improved either.