Behavioral Cloning

Introduction

This project helps to drive the car on its own by studying Human Behavior. The data is recorded from the simulator and then the model is trained with respective images and their steering angles. I used the dataset provided by Udacity. I did record images myself but when I train it with mine, there was a minor error by car going little out of track .The dataset contains JPG images of dimensions 160x320x3.

Augmentation Techniques Used

I have to thank this NVIDIA paper for suggesting these techniques.

Recovering using left and right camera images

I have added a small angle 0.25 to the left camera and subtract a small angle of 0.25 from the right camera. This is done to bring back the car to the center if it wanders off to the sides.

Flip the images horizontally

This is done to overcome the bias created during the training because the car turns to the left during most of the track and the images need to be flipped horizontally to simulate right turning and reverse steering angle.

Images Preprocessing

1. I noticed that the hood of the car is visible in the lower portion of the image and unwanted trees and skies in the image can be removed.
2. I have cropped 70 pixels from top and 25 from below to fasten processing of GPU.

Data Generation Techniques Used

Data is augmented and generated using python generators. So for every epoch, the optimizer practically sees a new and augmented data set. The above methods of flipping the images horizontally, preprocessing techniques of chopping out images and steering angle corrections are used for Dataset Generation. Yield generator is used for faster batch processing.

Model Architecture

The convolutional neural network architecture used is very similar to NVIDIA's End to End Learning for Self-Driving Cars paper. The main difference is that I used MaxPooling layers just after each Convolutional Layer in order to cut down training time.

Approach to choosing this Architecture

The hyperparamaters chosen for the model are the same as per mentioned in the NVIDIA network with 5 convolution layers and respective MaxPooling layers to avoid overfitting. I chose this model because I felt this model can cover almost every image recognition tasks and it is very much sophisticated with respect to design. I tried with batch sizes of 64 and 128 but I stuck with 64 at the end for good training. I tried with smaller convolution layered model with aggressive training strategies by adding a higher learning rate up to 0.1 and varied other parameters like filter size upto 8 and 10. It did not work. So, I stuck with original paper described network and it seemed to work very well. It is very elementary that I chose 85:15 ratios of training and validation.

Training

The data was taken from data folder from Udacity. I created two generators namely:

- training_generator = generate(train, batch_size = BATCH_SIZE)
- validation_generator = generate(valid, batch_size = BATCH_SIZE)

Batch size of both training_generator and validation_generator was 64. The training data and validation data are split in 85:15 ratio as per data available. I used Adam optimizer with 1e-4 learning rate. Finally, when it comes to the number of training epochs I tried several possibilities such as 5, 8, 10, 25 and 50. However, 10 works well on both training and validation tracks. The fit generator generates additional data from the preprocessing techniques used in above functions namely generate and get_images function.

Final Model Summary:

```
Layer (type)                     Output Shape          Param #     Connect
ed to
====================================================================================
==========================
lambda_1 (Lambda)                (None, 160, 320, 3)   0           lambda_
input_1[0][0]
```

cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0	lambda_1[0][0]
convolution2d_1 (Convolution2D)	(None, 33, 160, 24)	1824	cropping2d_1[0][0]
activation_1 (Activation)	(None, 33, 160, 24)	0	convolution2d_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 32, 159, 24)	0	activation_1[0][0]
convolution2d_2 (Convolution2D)	(None, 16, 80, 36)	21636	maxpooling2d_1[0][0]
activation_2 (Activation)	(None, 16, 80, 36)	0	convolution2d_2[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 15, 79, 36)	0	activation_2[0][0]
convolution2d_3 (Convolution2D)	(None, 8, 40, 48)	43248	maxpooling2d_2[0][0]
activation_3 (Activation)	(None, 8, 40, 48)	0	convolution2d_3[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 7, 39, 48)	0	activation_3[0][0]
convolution2d_4 (Convolution2D)	(None, 7, 39, 64)	27712	maxpooling2d_3[0][0]
activation_4 (Activation)	(None, 7, 39, 64)	0	convolution2d_4[0][0]
maxpooling2d_4 (MaxPooling2D)	(None, 6, 38, 64)	0	activation_4[0][0]

```
convolution2d_5 (Convolution2D)  (None, 6, 38, 64)     36928       maxpool
ing2d_4[0][0]
____________________________________________________________________________
____________________
activation_5 (Activation)        (None, 6, 38, 64)     0           convolu
tion2d_5[0][0]
____________________________________________________________________________
____________________
maxpooling2d_5 (MaxPooling2D)    (None, 5, 37, 64)     0           activat
ion_5[0][0]
____________________________________________________________________________
____________________
flatten_1 (Flatten)              (None, 11840)         0           maxpool
ing2d_5[0][0]
____________________________________________________________________________
____________________
dense_1 (Dense)                  (None, 1164)          13782924    flatten
_1[0][0]
____________________________________________________________________________
____________________
activation_6 (Activation)        (None, 1164)          0           dense_1
[0][0]
____________________________________________________________________________
____________________
dense_2 (Dense)                  (None, 100)           116500      activat
ion_6[0][0]
____________________________________________________________________________
____________________
activation_7 (Activation)        (None, 100)           0           dense_2
[0][0]
____________________________________________________________________________
____________________
dense_3 (Dense)                  (None, 50)            5050        activat
ion_7[0][0]
____________________________________________________________________________
____________________
activation_8 (Activation)        (None, 50)            0           dense_3
[0][0]
____________________________________________________________________________
____________________
dense_4 (Dense)                  (None, 10)            510         activat
ion_8[0][0]
____________________________________________________________________________
____________________
activation_9 (Activation)        (None, 10)            0           dense_4
[0][0]
____________________________________________________________________________
____________________
dense_5 (Dense)                  (None, 1)             11          activat
ion_9[0][0]
============================================================================
============================
Total params: 14,036,343
Trainable params: 14,036,343
```

```
Non-trainable params: 0
```

The above statistics mentions the layers in the model, the output shape when the layer functions with parameter have been applied, the number of parameters and the respective functions of whether it is convolution, maxpooling, flatten, lamda, activation or dense layers. The output shape clearly provides the number of parameters resulted in each layer and the input to the next layer of the network.

Results

In the initial stage of the project, I used a dataset generated by myself. That dataset was small and recorded while navigating the car using the laptop keyboard. However, the model built using that dataset was not good enough to autonomously navigate the car in the simulator. However, later I used the dataset published by the Udacity. The model developed using that dataset (with the help of augmented data) works well on both tracks as shown in following videos.