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 Kera's 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

Files Included:

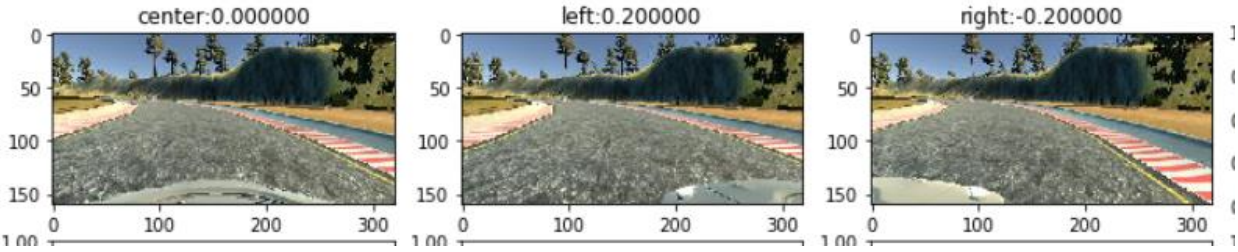
- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report.pdf

Model Architecture

- CNN built using NVIDIA's model as baseline
- Modifications include :-
    - Usage of ELU instead of RELU activation as it can produce negative outputs
    - Cropping of image to isolate only the road portion
    - Dropout before flattening to avoid overfitting

Loading and Augmentation

The dataset provided as the sample was used. The images were adjusted based on if they were center, left, or right images. Center images have a steering angle of 0 so flipping or angle adjustment is not required. Left and images have steering angles so flipping and angle adjustment of +0.2 for Left and -0.2 for Right images is required to avoid vehicle from going off track.



The additional data generated was stored in a dictionary generated_data. The aug_data function was used to make the adjustments. Finally, after iterating through every row in the driving log file the images were stored in a NumPy array X and the steering angles were stored in a NumPy array y.

Training the Model:

Using X and y as inputs with a 70-30 train test split the model was trained.

The model consists of the following layers:

Normalized Input Layer

Cropping Layer to select only required portion from image

5 Convolutional Layers with ELU activation as it gives negative outputs as well

A dropout layer to avoid overfitting model to training set data; Dropout value used is 0.5

4 Fully Connected Layers following Nvidia's model with 1164,100,50 and 10 neurons in each layer.

1 Output Layer

Parameter Tuning:

The model was compiled using the following parameters:-

MSE for loss as it is ideal for regression problems

Adam Optimizer

Learning Rate=1e-3

Accuracy for metrics

Final Architecture

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, 31, 158, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 77, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 37, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 35, 64)	27712
conv2d_5 (Conv2D)	(None, 1, 33, 64)	36928
dropout_1 (Dropout)	(None, 1, 33, 64)	0
flatten_1 (Flatten)	(None, 2112)	0
dense_1 (Dense)	(None, 1164)	2459532
dense_2 (Dense)	(None, 100)	116500
dense_3 (Dense)	(None, 50)	5050
dense_4 (Dense)	(None, 10)	510
dense_5 (Dense)	(None, 1)	11

Total params: 2,712,951
Trainable params: 2,712,951
Non-trainable params: 0

Finally, it was fitted on X and y (images and corresponding angles) for 5 epochs. Using drive.py and model.h5 the autonomous mode was tested and recorded as video.mp4.

References:

I applied concepts I had learnt from the course Convolutional Neural Networks by deeplearning.ai

Step by step Project Instructions provided by Udacity for this project