Udacity Self-Driving Car Engineer Nanodegree
Behavioral Cloning Project

Introduction

The goals / steps of this project are the following:

1. Use the simulator to collect data of good driving behavior
2. Build, a convolution neural network in Keras that predicts steering angles from images
3. Train and validate the model with a training and validation set
4. Test that the model successfully drives around track one without leaving the road
5. Summarize the results with a written report

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
- model.ipynb python notebook containing the code 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.md summarizing the results

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, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

I decided to test the model provided by NVIDIA as suggested by Udacity. I used the Keras lambda layer to normalize the data. The data was cropped to only reflect the road region. Below is the network architecture.

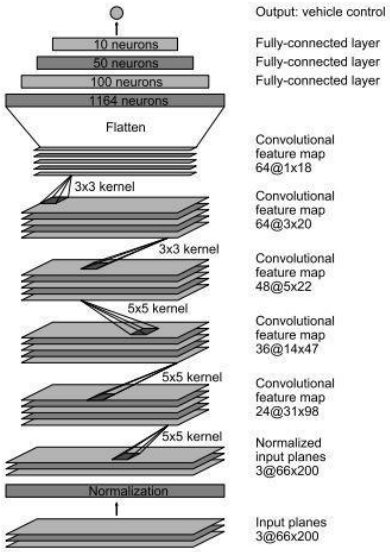
2. Attempts to reduce overfitting in the model
The model contains a dropout layer after the first fully connected layer with a dropout rate of 25%.

3. Model parameter tuning
No. of epochs = 5
Learning rate = 0.001
Generator batch size = 32
Optimizer = Adam

4. Appropriate training data
The data set was split in a 85%-15% ratio.
The data was flipped and augmented in order to familiarize the network about moving in the counter-clockwise direction too. The corresponding steering angles were hence negated. A correction factor of 0.2 in the steering angles was introduced. Hence, the left steering angle was increased by 0.2 and right was decreased by 0.2.

Model Architecture and Training Strategy

1. Solution Design Approach
My first step was to use a convolution neural network model similar to the one provided by NVIDIA autonomous team. I thought this model might be appropriate because it showed great output in the lecture. In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.
To combat the overfitting, I modified the model to incorporate a drop out layer with the drop out rate of 25% after the first fully connected. This helped my case and so no further actions were required.
The final step was to run the simulator to see how well the car was driving around track one. With the image pre-processing, training set augmentation and measures to reduce overfitting already done, my testing worked fine. I tuned my steering angle correction slightly from 0.25 to 0.2. The result was fine! 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 consisted of a convolution neural network with the following layers and layer sizes,

Layer (type)	Output Shape	Param #	Connected 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, 31, 158, 24)	1824	cropping2d_1[0][0]
activation_1 (Activation)	(None, 31, 158, 24)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 14, 77, 36)	21636	activation_1[0][0]
activation_2 (Activation)	(None, 14, 77, 36)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 5, 37, 48)	43248	activation_2[0][0]
activation_3 (Activation)	(None, 5, 37, 48)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 3, 35, 64)	27712	activation_3[0][0]
activation_4 (Activation)	(None, 3, 35, 64)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 1, 33, 64)	36928	activation_4[0][0]
activation_5 (Activation)	(None, 1, 33, 64)	0	convolution2d_5[0][0]
flatten_1 (Flatten)	(None, 2112)	0	activation_5[0][0]
dense_1 (Dense)	(None, 100)	211300	flatten_1[0][0]
activation_6 (Activation)	(None, 100)	0	dense_1[0][0]
dropout_1 (Dropout)	(None, 100)	0	activation_6[0][0]
dense_2 (Dense)	(None, 50)	5050	dropout_1[0][0]
activation_7 (Activation)	(None, 50)	0	dense_2[0][0]
dense_3 (Dense)	(None, 10)	510	activation_7[0][0]
activation_8 (Activation)	(None, 10)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	11	activation_8[0][0]

I used the ELU activation instead of RELU as I read they are work better and I saw that in my architecture.

3. Creation of the Training Set & Training Process

I tried to take the training data as per suggestions given in the classroom but being bad at maneuvering the car properly and not obtaining correct results, I decided to go for the sample data set given by Udacity.