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

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 (the script is the same as one in model.py)
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_BehavioralCloning.md and writeup_BehavioralCloning.pdf 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

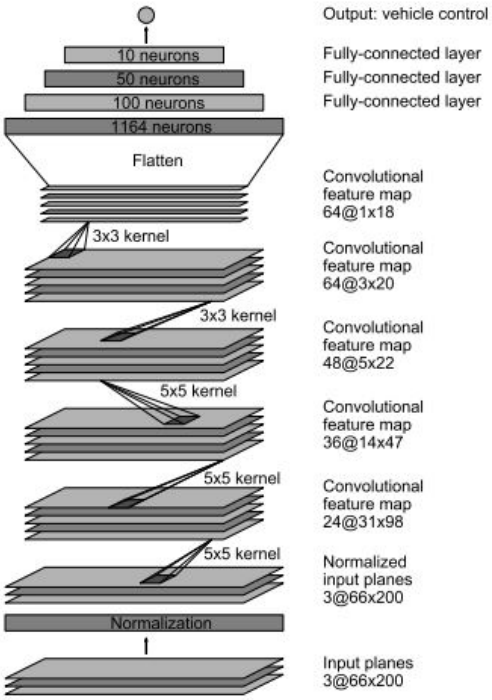
The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

A model provided by NVIDIA as suggested by Udacity is applied in this project. The model architecture is described by NVIDIA here. As an input this model takes in image of the shape (60,266,3) but our dashboard images/training images are of size (160,320,3). I keep the architecture of the remaining model same but feed an image of different input shape.

The following is the model architecture.



The proposed model consists of a convolution neural network with 5x5 filter sizes for three layers and 3x3 filter sizes for two layers with depths between 18 and 200 (model.py lines 163-181)

The model includes ELU layers to introduce nonlinearity (code line 165), and the data is normalized in the model using a Keras lambda layer (code line 158).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 191).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 142-143). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

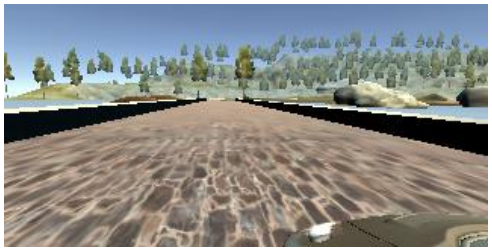
- The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 209).
- No of epochs= 5
- Learning Rate- Default 0.001
- Validation Data split- 0.15
- Generator batch size= 32
- Correction factor- 0.2
- Loss Function Used- MSE(Mean Squared Error as it is efficient for regression problem).

4. Appropriate training data

- The dataset is provided by Udacity (also can be obtained from AWS). The trained mode is applied to collect training data. However, it takes considerable time to do data collection. Finally, the dataset provided by Udacity is applied.
- I am using OpenCV to load the images, by default the images are read by OpenCV in BGR format but we need to convert to RGB as in drive.py it is processed in RGB format.
- Since we have a steering angle associated with three images we introduce a correction factor for left and right images since the steering angle is captured by the center angle.
- A correction factor of 0.2 is applied.

- For the left images I increase the steering angle by 0.2 and for the right images I decrease the steering angle by 0.2
- Sample Images provided by Udacity and collected by myself.Sample Input Image from Udacity

Sample Input Image from Udacity



Sample Input Image Collected by Myself

Model Architecture and Training Strategy

1. Final Model Architecture

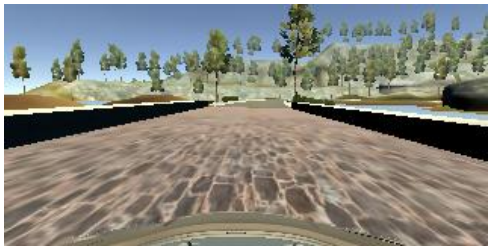
- The final model architecture (model.py lines 150-223) consisted of a convolution neural network with 5x5 filter sizes for three layers and 3x3 filter sizes for two layers with depths between 18 and 200 (model.py lines 163-181).

The following is the final model architecture.

```
Epoch 1/5
6830/6830 [==============================] - 75948s 11s/step - loss: 0.0126 - val_loss: 0.0180
Epoch 2/5
6830/6830 [==============================] - 9196s 1s/step - loss: 0.0051 - val_loss: 0.0165
Epoch 3/5
6830/6830 [==============================] - 31548s 5s/step - loss: 0.0029 - val_loss: 0.0155
Epoch 4/5
6830/6830 [==============================] - 8518s 1s/step - loss: 0.0021 - val_loss: 0.0148
Epoch 5/5
6830/6830 [==============================] - 36138s 5s/step - loss: 0.0016 - val_loss: 0.0150
Done! Model Saved!
_________________________________________________________________
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
_________________________________________________________________
activation_1 (Activation)    (None, 31, 158, 24)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 77, 36)        21636
_________________________________________________________________
activation_2 (Activation)    (None, 14, 77, 36)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 37, 48)         43248
_________________________________________________________________
activation_3 (Activation)    (None, 5, 37, 48)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 35, 64)         27712
_________________________________________________________________
activation_4 (Activation)    (None, 3, 35, 64)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 33, 64)         36928
_________________________________________________________________
activation_5 (Activation)    (None, 1, 33, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2112)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               211300
_________________________________________________________________
activation_6 (Activation)    (None, 100)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
activation_7 (Activation)    (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
activation_8 (Activation)    (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 348,219
Trainable params: 348,219
Non-trainable params: 0
_________________________________________________________________
```

- As it is clear from the model summary the first step is to apply normalization to the all the images.
- Second step is to crop the image 70 pixels from top and 25 pixels from bottom. The image was cropped from top because I did not wanted to distract the model with trees and sky and 25 pixels from the bottom so as to remove the dashboard that is coming in the images.

Sample Input Image-



Cropped Image-

- Next Step is to define the first convolutional layer with filter depth as 24 and filter size as (5,5) with (2,2) stride followed by ELU activation function
- Moving on to the second convolutional layer with filter depth as 36 and filter size as (5,5) with (2,2) stride followed by ELU activation function
- The third convolutional layer with filter depth as 48 and filter size as (5,5) with (2,2) stride followed by ELU activation function
- Next we define two convolutional layer with filter depth as 64 and filter size as (3,3) and (1,1) stride followed by ELU activation funciton
- Next step is to flatten the output from 2D to side by side
- Here we apply first fully connected layer with 100 outputs
- Here is the first time when we introduce Dropout with Dropout rate as 0.25 to combact overfitting
- Next we introduce second fully connected layer with 50 outputs
- Then comes a third connected layer with 10 outputs
- And finally the layer with one output.

2. Creation of the Training Set & Training Process

- The Udacity Dataset is analyzed and it is found out that it contains 9 laps of track 1 with recovery data.
- The dataset is split into training and validation set using sklearn preprocessing library.
- Keep 15% of the data in validation Set and the remaining is in training Set
- Use generator to generate the data so as to avoid loading all the images in the memory and instead generate it at the run time in batches of 32. Even Augmented images are generated inside the generators.

Output Video

Output video run1.mp4 is included in delivery package.

Important SideNotes-

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track. To improve the driving behavior in these cases, I plan to do more tests to tune in the parameters to have more smoother driving.

The workspace sometimes is not stable. I have the issues to run the "python drive.py model.h5" sometimes. I will take more time to solve this problem.