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
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report.pdf summarizing the results
- video.mp4 containing recording of vehicle driving autonomously around the track
- model.ipynb containing steering angles histogram and samples of images used

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 have used two separate models during training:

- Lenet convolution neural network
- Nvidia CNN

A/ Lenet

```
Input: 160x320x3 (cropped on the vertical size to various values)
Conv: (5x5x3) filter with stride 1
Relu
Maxpool: (2x2) kernel, (2x2) stride
Conv: (5x5x6) filter with stride 1
Relu
Maxpool: (2x2) kernel, (2x2) stride
Fully Connected. Input = 400. Output = 120
Relu
Fully Connected. Input = 120. Output = 84
Relu
Fully Connected. Input = 84. Output = 10
```

The LeNet model consists of a convolution neural network with 5x5 filter sizes and depths between 6 and 16

B/ Nvidia

```
Input: 160x320x3 (cropped on the vertical size to various values)
3 Conv layers, : (5x5x3) filter with stride, with Relu and Maxpool: (2x2) kernel,
(2x2) stride
2 Conv layers: (3x3x3 1) filter with stride 1 with Relu and Maxpool: (2x2) kernel,
(2x2) stride
Fully Connected. Input = 400. Output = 120 with Relu
Fully Connected. Input = 120. Output = 84 with Relu
Fully Connected. Input = 84. Output = 10 with relu
Fully Connected. Input = 10. Output = 1
```

The Nvidia model consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 64

Both models includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer.

Since the model output a single continuous numeric value, one appropriate error metric would be mean squared error.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting.

The model was trained and validated on different data sets to ensure that the model was not overfitting. Initially, Keras selected the validation test automatically when model.fit was used (configured to 20%), then I created it explicitly (also at 20%), when stearted using model.fit_generator (along with a generator)

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 25).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving and recovering from the left and right sides of the road. I created a datasets with 10 laps on track 1. The first 9 laps were center lane driving, the 10th lap included recoverings. However, the results I got were worse than with the original data set provided, so I ended up sticking with the sample dataset.

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

I started with the Letnet architecture. 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 in several ways:

- I introduced augmented measurements by flipping each each image and associating a reversed steering angle, effectively doubling the dataset size. I continued to see overfitting, as validation loss inititally decayed but then started to grow again after the 4th epoch
- I then introduced dropout, but overfitting continued to occur. I did not use dropout in any of the subsequent LeNet configurations
- I then included images from left and right cameras. I tried several correction values, between 0.1 and 0.5, best results were obtained with 0.125. This enabled me to trippled the dataset (now 6 times larger than the original dataset)
- I then tried cropped images. I tried both the original suggestion of (70,10) on the vertical axis, i.e., dropping to the 70 pixels and the bottom 10, but I actually got worse results than w/o cropping.

I also created a generator to be able to efficiently train in memory with even larger datasets. Did not see any difference on results after implementing the generator.

After trying all these options on the Lenet architecture, I tried a more powerful network, the one from the Nvidia paper. I started with the latest configuration from the Letnet model, namely, no dropout, no cropping, and with both augmented images and left/right images. Initial results did not show overfitting, because both the training and validation loss decreased across all 6 epochs. I then added cropping and dropout (with 0.3 probability), and results improved, while overfitting still not ocurring.

These are the best results obtained:

```
Epoch 1/6
38568/38568 [==============================] - 76s - loss: 0.0156 - val_loss:
0.0135
Epoch 2/6
38568/38568 [==============================] - 71s - loss: 0.0127 - val_loss:
0.0124
Epoch 3/6
38568/38568 [==============================] - 72s - loss: 0.0123 - val_loss:
0.0121
Epoch 4/6
38568/38568 [==============================] - 72s - loss: 0.0119 - val_loss:
0.0120
Epoch 5/6
38568/38568 [==============================] - 72s - loss: 0.0118 - val_loss:
0.0117
Epoch 6/6
38568/38568 [==============================] - 72s - loss: 0.0116 - val_loss:
0.0115
```

(I trained on a g2.xlarge AWS instance)

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. My best result was to cross the brige, but then after the bridge the car hit the dirt instead of staying on the track.

I have not been able to go beyond the dirt after the bridge.

2. Final Model Architecture

The final model architecture (model.py lines 111-124) is the Nvidia architecture as described in the previous section, with the configuration also described in the previous section.

This is the situation I am in:

- there is no overfitting in my results (both training and validation loss decreased in all 6 epochs)
- I have tried most of the improvements suggested (flipped image, left/right camera, cropping, dropout)
- More epochs I do not think will bring much benefit because loss variations between epochs are very small

This leads me to believe I have a good enough network, and that the next improvement would come from having a better dataset, specially one that is able to have enough samples showing how to avoid the dirt part of the track. Given that I have a generator implemented, and I am training on g2.xlarge AWS instance, I believe also should be able to handle a larger dataset.

Again, I tried with tha 10 lap dataset, but only the last lap contained recovering steps. And it also included the drifting step, which as mentioned in the course should be avoided, in order to have the network learn only recovery, and not drifting.

Update based on feedback from reviewer:

- I have created a new model.ipynb notebook to plot steering angles' histogram and sample images
- I removed samples with steering angle = 0 using the following code within my generator fuction (lines 37-38 model.py):

```
if measurement == 0:
    continue
```

However, results were worse than when considering them

- I have increased the number of epochs to 10, but my AWS g2.xlarge instance with Udacity AMI stopped detecting the GPU for some reason, and started working on the CPU. This makes training *much* slower. This is the error:

```
modprobe: ERROR: could not insert 'nvidia_375_uvm': No such device
E tensorflow/stream_executor/cuda/cuda_driver.cc:509] failed call to cuInit:
CUDA_ERROR_UNKNOWN
I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:145] kernel driver does not
appear to be running on this host (ip-10-89-19-89): /proc/driver/nvidia/version
does not exist
```

It seems I would need to create a new instance from scratch.