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

Rubric Points

Each of the rubric points required for this project are listed below with an explanation of how the particular point is met.

Files Submitted

Submission includes all required files and can be used to run the simulator in autonomous mode on track 1 (track on left hand side of the simulator track options). It includes the following files:

- model.py containing the script to create and train the model
- drive.feb28.nvidia.py for driving the car in autonomous mode
- model.nvidia20Ephochs.h5 containing a trained convolution neural network
- res.mp4 capturing the vehicle driving autonomously around the track, slight more then one time. The video is from the center camera image viewpoint.
- writeup_report.pdf summarizing the results

Quality of Code:

- The “model.py” code is fully functional and can be used to train the car to drive by running:

    python model.py

- This script stops at designated points to allow the user to explore python data structures as necessary. To continue the script the user enters “Ctrl D”, until the script finishes.

- The trained model can then be successfully driven on track 1 of the Udacity simulator without crashing, or going off the road, even temporarily, by performing the following command:

    python drive.feb28.nvidia.py model.nvidia20Ephochs.h5

- Please note - The car was driven in the simulator autononmously for many hours.

- The training model does make use of a python generator to generate data images for training rather than storing the data images in the memory.

- Comments are provided in the code.

Model Architecture and Training Strategy

- An appropriate model architecture has been employed:

A training model written in Keras (& using Tensorflow), based on the published nvidia model (link below) was used to train the model. This model includes several convolutional layers, followed by several fully connected layers. The data was normalized within the model using a keras lambda function. The implementation of this model can be found in the model.py lines 237-278. Further information is provided in the next section.

https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf was used.

- Train and validation test splits (80/20) of the data were used.

- However, to stay true to the nvidia model, drop out was not used. Instead, additional data images were generated with randomnous during the training so as to emulate a much larger training set then was actually available. Note: that even though there were just over 6000 images used in each

epoch of training, that the images used per epoch were not exactly the same set of images in each epoch.

- The model was exclusively trained & validated with the training images provided by Udacity but additional data augmentation was used, to essentially expand the data set.
    - All 3 images (left, center, right) were randomly selected and if the image was from the left or the right, then the steering angle was adjusted by 0.25 or -0.25 respectively. (see model.py lines 144-156)
    - Images were cropped to remove unnecessary data from the image (such as the car hood and the sky) by removing the top 1/5 (32 pixels) and bottom 25 pixels from the original image to convert the original 160 x 320 image to 103 x 320 image. (see model.py lines 79-91)
    - Then the image was re-sized to adhere to the nvidia model to be 66x200 (height x width), so that when the image entered the nvidia convolutional layers it was as per what they had documented. (see model.py lines 79-91)
    - Once the images were cropped, then the images were randomly flipped horizontally and the steering angle adjusted accordingly. This was done to create more images of the car going in a different direction on the track. (see model.py lines 93-102)
    - An additional augmentation of shifting images randomly vertically and horizontally was created and tested to shift properly but was not incorporated in the training augmentation as the model was able to drive autonomously on track 1 without it. (This procedure is left in the code but is not used and will be explored in future work by the author for the 2nd track)
- An “adam” optimizer was used, so the learning rate was not tuned manually. It was set with an initial learning rate of lr=0.0001. (model.py line 352)
- A mse (mean squared error) graph per training is produced to examine the extent of the possible overfitttting based on the training vs validation for each training run. (see model.py lines 60-70)

- The model was trained for various number of ephochs (5, 10, 20), and was able to navigate track 1 in the simulator without crashing. While training for 5 & 10 epochs resulted in the car being able to circulate the track successfully, it still occasionally would go up on the ledges but recovered. At 20 epochs the car stayed within the roadway at all times and did not go up on the edge of the track. The model was run for several hours and did not crash.

- In order to run the model in the simulator, it was necessary to do the same pre-processing of the images given by the simulator as those provided for training such that the images given to the model to predict the steering angle were also the nvidia-sized images 66x200, which is what the trained model expected. Thus a copy of drive.py was made to drive.feb28.nvidia.py in which pre-processing of images was called (see drive.feb28.nvidia.py line 67, and model.py lines 79-91)

- To develop the code for the model, several “helper” procedures were developed to display one image (see lines model.py lines 42-50) and to display a histogram of the steering values for the provided training data (see model.py lines 53-55). The display one image procedure was used to insure that the various data augmentation routines were working properly. Another routine was also developed to print out the model (see model.py lines 300-307).

- One additional strategy for development was used - Important to Start Simple! And add functions incrementally!

    - For the initial development, a very simple model was created (see model.py 570-582) that did the cropping (removal of sky and car hood) of the image using a Keras layer in the model, and normalization within the model. This model only had one simple fully connected layer of 128 with RELU non-linearity activation. It was trained for just one epochs at first on the center images of the Udacity training data, with no additional augmentation being performed. It also made use of a python generator to pass the training data to the model. In the simulator this model was able to drive the car but typically crashed on the bridge.

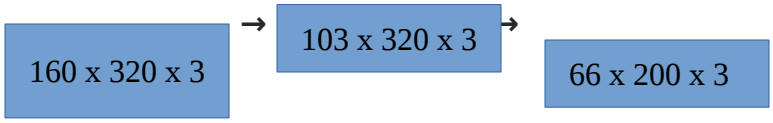
    - Then, additional pre-processing was integrated, but this had to be done outside of the Keras model. This was done by introducing one pre-processing element at a time, training and seeing the impact. Once that

was working the model was trained for 5-10 epochs and managed to get across the bridge but would crash.

- The the last few layers of the nvidia model, the fully connected layers, were added and the activation function was changed from RELU to ELU (with the relu's as activation it seemed to be making the model only generate 0's.)
- Then, finally all the nvidia keras layers were added and the final results obtained.

Model Architecture

INPUT To NVIDIA Model:



NVIDIA MODEL