Behavioral Cloning Project

For this project, I trained a convolutional neural network model to simulate an autonomous driving car. The model that is being submitted drives a car so that it does not leave the track and endanger the driver. Below are the summary of the steps taken to complete the project.

Files Submitted & Code Quality

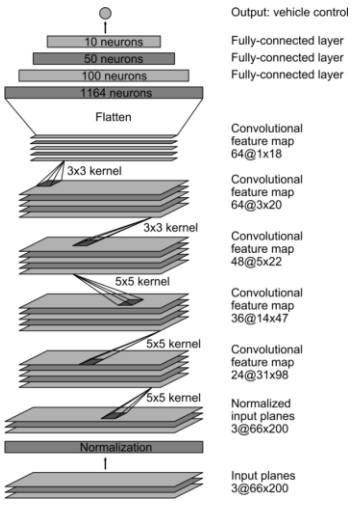
1. **Submission includes all required files and can be used to run the simulator in autonomous mode**
   My project includes the following files:
   - model.py contains the script to create and train the model
   - drive.py for driving the car in autonomous mode
   - model.h5 contains a trained convolution neural network
   - writeup_report.pdf summarizes the results
   - model_sim.mp4 is the video of the simulated autonomous driving by the model
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**

Model Architecture

The model is copied from Nvidia's self-driving car architecture and is made up of 1 normalization layer, 5 convolutional layers, and 3 fully connected layers for a total of 9 layers (Figure 1). Here are some brief notes about the model.

- To reduce overfitting, I have included dropouts in the fully connected layers this this architecture.
- The model includes RELU activation functions to introduce nonlinearity.
- Dataset is split into 80% training data and 20% validation data.
- Training dataset is augmented with flipped images.
- Data is normalized using a Keras lambda layer.
- The final model uses an adam optimizer with a learning rate of 0.0001 and was trained for 15 epochs.
- It uses generators to reduce the memory space and quicken the training time.

Figure 1: Nvidia's self-driving car network architecture



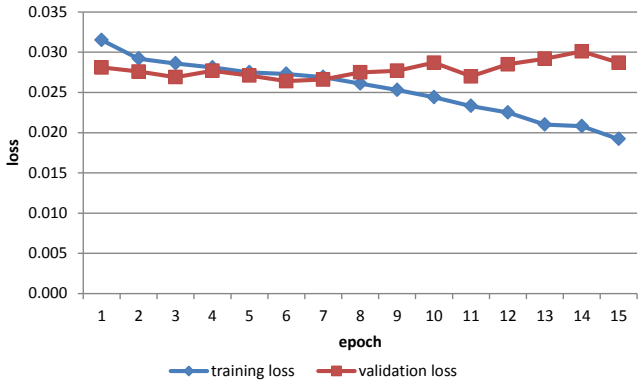
Source: https://devblogs.nvidia.com/deep-learning-self-driving-cars/

Methodology

Here are steps that were taken to come to the final model. I do want to point out that there were multiple iterations in between each of the steps highlighted below and this is just a condensed version.

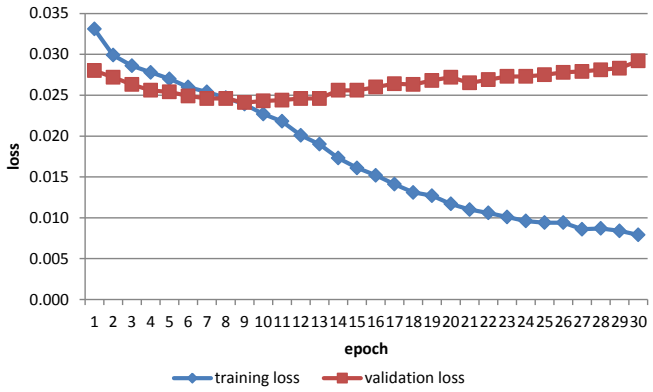
- I initially started out with driving data/images provided by Udacity. It was split into 80% training and 20% validation dataset and the training dataset was augmented with flipped images. It ran on Nvidia network architecture for 5 epochs using the default adam optimizer. The result was not satisfactory.
- I then added dropouts in fully connected layers to reduce overfitting. The result was still not satisfactory.
- I then added generator functionality to the code. Initially, I removed the augmentation of the training dataset. This reduced each epoch time from roughly 800 seconds to 12 seconds. While time was saved, result was still not satisfactory.
- I included back the augmentation of the training dataset. This increased each epoch time to roughly 15 seconds. Result was not satisfactory.
- I trained more data on my own using the simulator. While doing this, I tried to add data of extreme cases where the car starts from the outer edge of the screen off from the road and drive back into the road. Meanwhile, I was fearful that I might introduce bad data because my simulated driving skill was not up to par as well as due to self-diagnosed carpal tunnel syndrome. After several days of using the simulator to record new data, I was able to increase the number of center camera images from 8,036 to 13,948. The epoch time increased to roughly 45 seconds due to increased dataset. The time spent to train new data was worth it because the resulting model drove better than me. The car stayed in the track boundary when using this new model.
- Given the momentum, I decided to keep on going to further improve the model. I increased the number of epoch from 5 to 15, and tried to observe whether I am training too few epochs or too much. As seen in Figure 2, validation loss was lowest at epoch number 6. So I re-ran with number of epoch to be 6. It was a big disappointment that this was even worse than what I ran with 5 epochs. Momentum was crushed.

Figure 2: Losses at each epoch point, learning rate = 0.001



- I then played around with learning rate of the adam optimizer. The default learning rate is 0.001. Figure 3 shows the losses at each epoch point when the learning rate is 0.0001 for 30 epochs.

  Figure 3: Losses at each epoch point, learning rate = 0.0001



- Final submitted model uses learning rate of 0.0001 with number of epoch of 15.

Areas of improvements to be made

- In my opinion, the biggest improvement to the model is to get more clean training data. Just adding some training data improved my result drastically. I believe this will be the biggest contributor to the model.
- In same regard, I could have used more augmentations of data. For example, I could have used images by shearing, cropping and changing brightness. In addition, there are images from three cameras. I could have also used images from left and right cameras for training. These will increase the data/image that can be trained on and are basically extensions of the previous bullet point.

As always, thank you for reviewing my project!