Udacity SDCND Project 3

Behavioral Cloning

Jesse R

Project overview

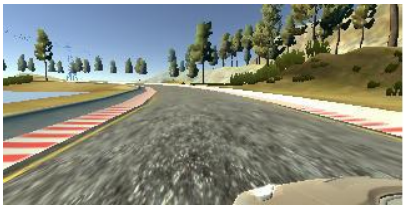
This project demonstrates the ability of a deep neural network to learn to predict the steering angle of a human driver based on images and steering angle collected during training sessions on a driving simulator. The goal is to clone the behavior of the driver and use the model to control the driving simulator in real time allowing it to make consistent laps around the track without collisions or leaving the course.

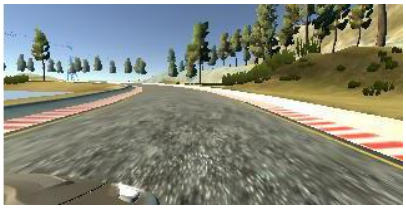
Data collection

The simulator has a recording mode for data collection. This mode stores virtual camera images throughout the training sessions from three cameras mounted to the simulated car. It also generates a log file of the steering angle and speed.

To get enough data for training the model I chose to record about 20 laps driving the course as close to center as possible. This was done with an analog gaming wheel which produced a much smoother input than the keyboard controls or the ps3 controller that I tested. I also recorded four laps in the opposite track direction as well as four laps driving the left side of the track and four more on the right side.

These are samples of the three camera views

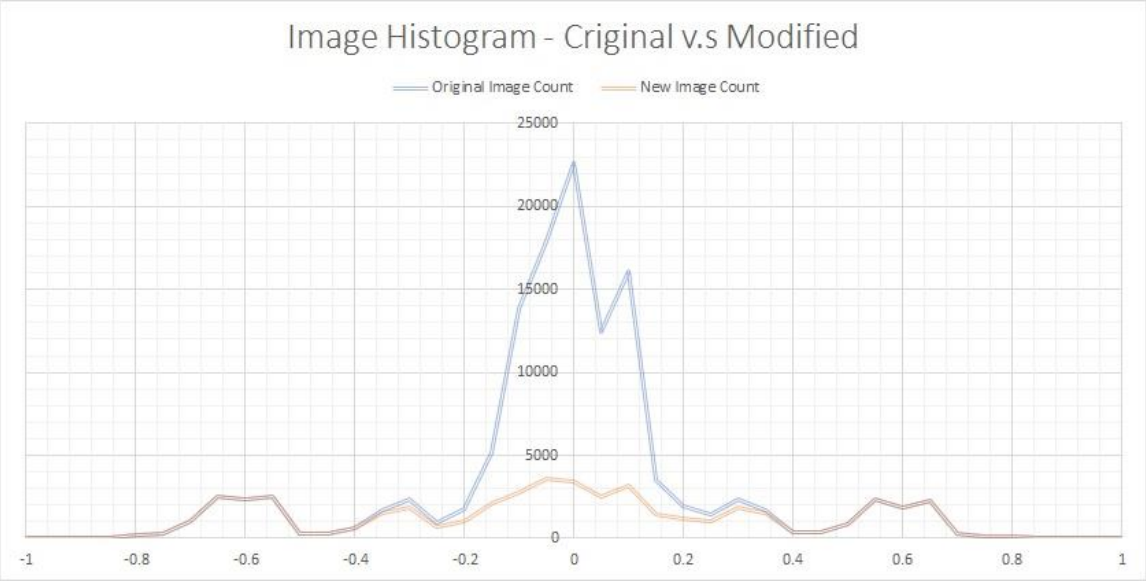




Data selection

As was done in the Nvidia project (https://arxiv.org/pdf/1604.07316.pdf), I chose to remove much of the uninteresting data along the straight sections of the track to allow the model to better fit to challenging corners. Of the original 124,383 images I chose to use 48,246 of the images by randomly removing data points that were close to zero steering angle.

This is a distribution of steering angles across the data before and after removing the low angle data.



I added an offset to each camera to give a right steering bias to images taken from the left side and vice versa. I tried several combinations but had the best results with 2 degrees correction for the left and right images. I also corrected the angle for the laps that were driven on the left and right sides of the track. The extreme outside images were adjusted by 16 degrees, the center was adjusted by 8 degrees and the inside camera was offset 2 degrees. Again, there were several combinations tested and this provided the best result.

Data Augmentation

To reduce overfitting I generated augmented data using a combination of three methods

- Image mirroring using a 50% random chance – I also negated the steering angle data when flipping the images
- Shifting the images left and right a random amount up to 25 pixels. – I added a correction factor to the steering angle to compensate (the model crops out the left and right 25 pixels to avoid 'seeing'the black bands that occur in the translation of the image.
- Image brightness – random adjustments to each color channel in the range of .8 to 1.2

The data was loaded via a generator into the Keras model described below.

Model Architecture

Several models were tested starting with a single dense layer attached to the output. I progressively added more layers, both CNN and dense until the model could achieve a low loss on the training data. In the end I chose to use a model very similar to the one in the Nvidia paper mentioned earlier. The key differences I used were

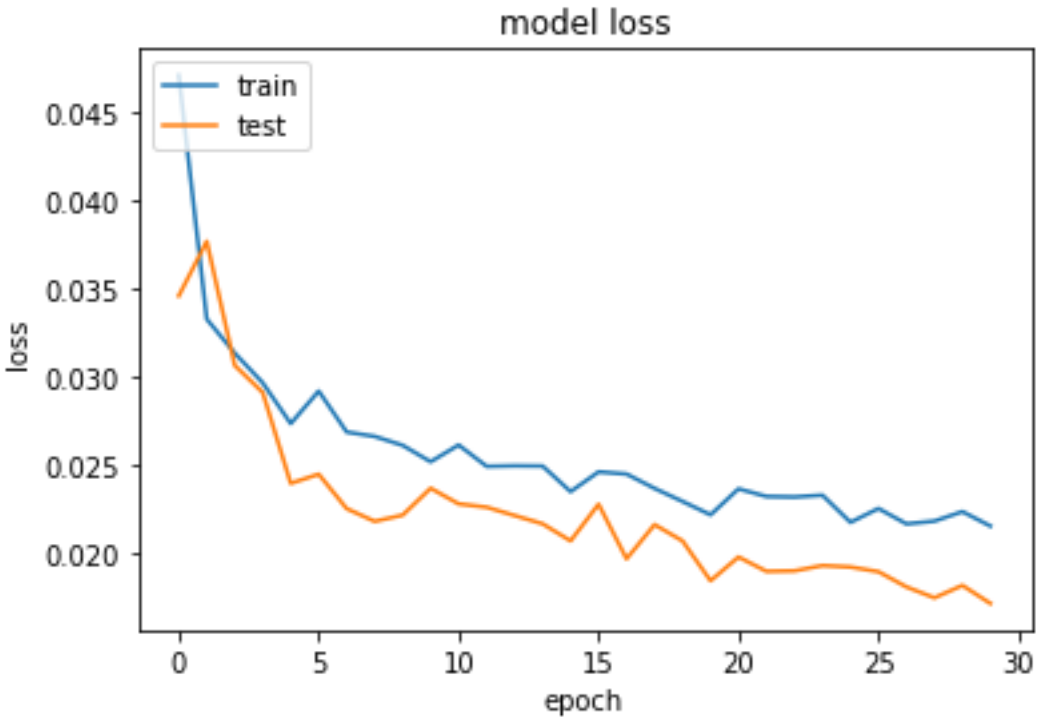
- 3x3 kernels in all the CNN layers
- Image input size was larger
- Used RELU for all layers
- 20% dropout on all layers
- All other model dimensions were the same (4 CNN layers, 3 fully connected and one output)

The data was preprocessed with Keras using a lambda layer to normalize the data to -.5 to +.5. The images were also cropped to remove the sky, the hood of the car and the left and right 25 pixels to avoid training on the black areas of the translated images. The model summary is below.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 160, 320, 3)       0
_________________________________________________________________
cropping2d_1 (Cropping2D)    (None, 65, 270, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 63, 268, 24)       672
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 134, 24)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 31, 134, 24)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 29, 132, 36)       7812
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 66, 36)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 66, 36)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 64, 48)        15600
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 32, 48)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 32, 48)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 30, 64)         27712
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 15, 64)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2, 15, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1920)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1164)              2236044
_________________________________________________________________
dropout_5 (Dropout)          (None, 1164)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               116500
_________________________________________________________________
dropout_6 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_7 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 2,409,441
Trainable params: 2,409,441
Non-trainable params: 0
_________________________________________________________________
```

Training

I trained the model several times with various settings eventually settling on the Adam optimizer to reduce the mean squared error using all data in each of the 8 Epochs. I tested the model with fewer and more epochs and could achieve better validation loss but the best model performance on the track seemed to be around the 8 epoch mark. As mentioned above, 20% dropout probability was used on each layer. Testing was done with and without dropout and at various keep probabilities and 20% produced the best results for me. This is a sample of the training loss across 30 mini batches. This is equivalent to about 6 actual epochs. The test error is better than the train error due to the way Keras handles the forward pass during validation, all dropouts are set to 100% on.



Results

After training at least 50 variations of the model and combinations of data sets and augmentation parameters I was able to get two different models that could navigate the track indefinitely at the maximum 30mph. Most models could complete the track at 9mph which seemed to be the minimum goal of this assignment. The two best models were not very graceful at 30mph but looked very good at speeds under 25mph. Sample runs can be seen at https://youtu.be/Zk2s65BEDpo and https://youtu.be/FXeJIIleblM