Behavioral Cloning

Write up of work done for Udacity CarND Behavioral Cloning Project P3, by Mani Srinivasan

---

Project Objectives

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

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

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
- Behavior cloning - P3 -Submit.ipynb, the Jypyter notebook
- writeup_report.pdf (this file) 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.json
```

3. Submission code is usable and readable

The model.ipynb is the Jupyter notebook that 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. I have also added the model.py code for sake of completion.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 3x3 filter sizes and depths between 32 and 128 (model.py lines 139-169)

The model includes ELU layers to introduce nonlinearity (143-168), and the data is normalized in the model using a Keras lambda layer (code line 140).

2. Attempts to reduce overfitting in the model

The model contains dropout layers to reduce overfitting (model.py lines 143-168).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code lines 88-104). 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, with a learning rate of 0.0001 (model.py lines 170-173).

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.

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

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to get the car to drive around track 1 w/o overfitting.

My first step was to use a convolution neural network model like the one in an article by NVidia. http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf. But having worked on it for close to a week and not finding a good solution, I switched to a CNN model described in another paper by Vivek Yadav, in https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9#.1g3gw99kj... This model immediately started yielding the results I was struggling to get initially. It uses a set of filters and convolutional networks, Maxpooling, dropout layers and fully connected layers. The first layer is 3 1X1 filter, which has the effect of transforming the color space of the images. Using 3 1X1 filters allows the model to choose its best color space. This is followed by 3 convolutional blocks each comprised of 32, 64 and 128 filters of size 3X3. These convolution layers were followed by 3 fully connected layers. All the convolution blocks and the 2 following fully connected layers had exponential relu (ELU) as activation function.

To gauge how well the model was working, I split my image and steering angle data into a training and validation set. I used 25% of the samples as validation data. To address overfitting, I used several image processing techniques including flipping, image augmentation by varying the brightness randomly, shifting the images to the left and right by a few pixels, using alternate camera images with modified steering angle etc.

It took a while to get the car to go around the track without falling off the edges. Until the car went around track 1 for a few laps, I did not add any data on my own.

At the end of the process, the vehicle was able to drive autonomously around the track without leaving the road. As the weaving of the car was a little disconcerting, I modified the drive.py by averaging the steering angle over multiple frames and also lowering the speed so that the car did not veer off course.

2. Final Model Architecture

The final model architecture (model.py lines 143-168) consisted of a convolution neural network with the following layers and layer sizes.

```
- Normalization Layer using Lambda function.
- 3 Sets of CNN layers with Maxpooling and dropout to prevent overfitting
  - Set 1:
    Convolution - 32x3x3 followed by ELU activation
    Convolution - 32x3x3 followed by ELU activation
    Maxpooling - 2x2 filter
    Dropout - 50%

  - Set 2:
    Convolution - 64x3x3 followed by ELU activation
    Convolution - 64x3x3 followed by ELU activation
    Maxpooling - 2x2 filter
    Dropout - 50%

  - Set 3:
    Convolution - 128x3x3 followed by ELU activation
    Convolution - 128x3x3 followed by ELU activation
    Maxpooling - 2x2 filter
    Dropout - 50%

- Fully Connected Layers
   FC 512 with ELU activation
   FC 64 with ELU activation
   FC 16 with ELU activation
   FC 1 with ELU activation (Output)
```

3. Creation of the Training and validation Set

For a large part of the work, I used the Udacity provided data to train and model the network. This is due to the difficulty I faced in controlling the car with a normal mouse. Later, to capture good driving behavior, I first recorded two laps on track 1 using center lane driving. I then recorded the vehicle recovering from the left and right sides of the road back to center so that the vehicle would learn to recover from edges without falling off.

The following sample videos demonstrate the recovery images collected.

https://youtu.be/2dOlrs80_YU

https://youtu.be/osb-lKCwYUA

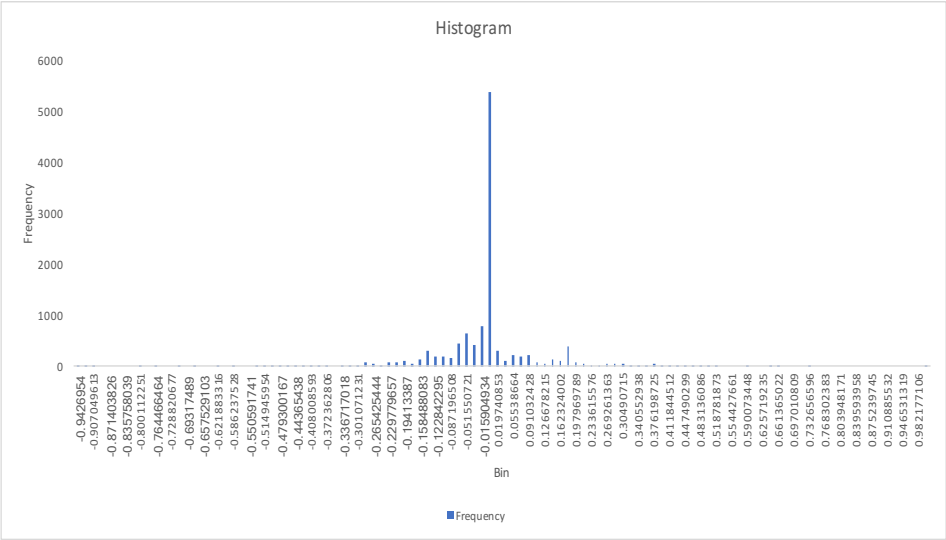
https://youtu.be/vVuPe2Tr0TI

After the collection process, I had about 11540 data points. I would have collected more to get a much smoother drive but for the difficulty in controlling the car with the mouse, adjusting the speed and recording at the same time on my MacBook Pro.

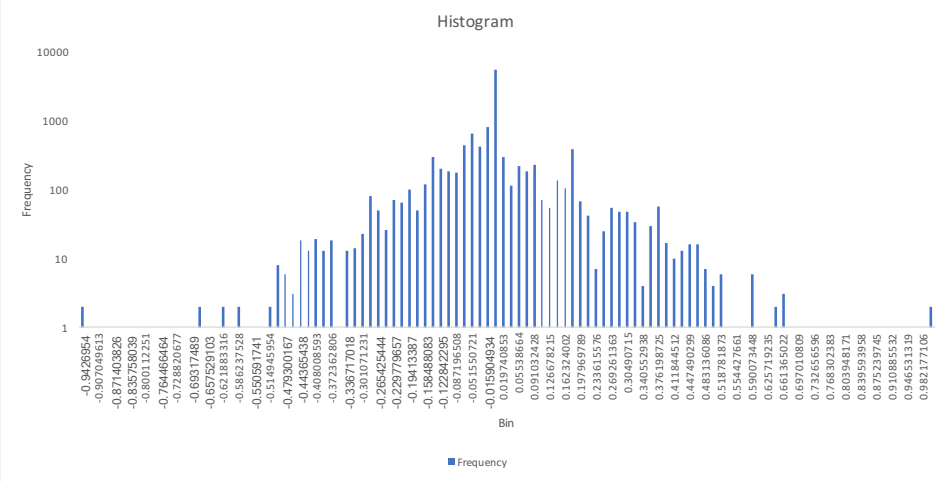
I finally randomly shuffled the data set and put 25% of the data into a validation set.

4. Data Augmentation Techniques

As the dataset was limited in size and had more left turns than right turns on track 1 (see the histogram below) I had used some data augmentation techniques as discussed in the next few paras to prevent over-fitting of the data. Augmentation is the process of generating new training data from a smaller data set to represent real world scenarios rather than collecting a huge data set which is difficult if not impossible due to the capture and storage efforts involved. Augmentation helps in manipulating the incoming training data to generate more instances of training data and has been used to develop powerful classifiers with little data.



Histogram Plot of steering angles



Logarithmic Plot

4.1. Brightness Augmentation

Brightness augmentation or changing brightness is a technique used to simulate day and night conditions and ensures not to over-fit the data, as track 1 has a bright scenario throughout. The images were adjusted with different brightness by first converting them to HSV, scaling up or down the V channel and converting the images back to RGB.

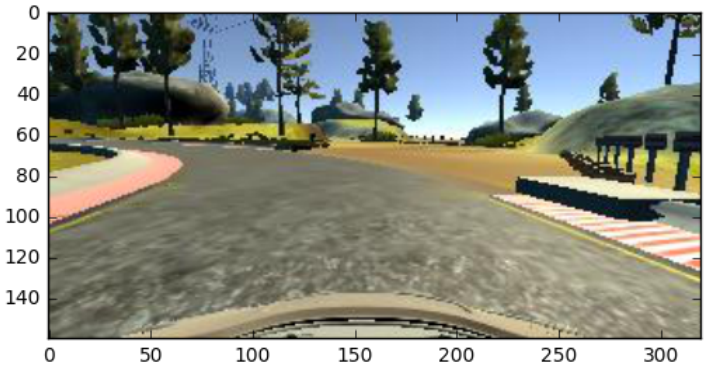


Image without brightness augmentation

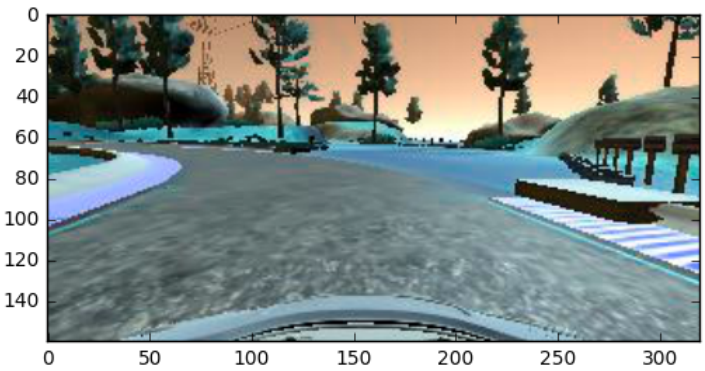


Image with brightness augmentation

4.3. Image Flipping

In addition to the transformations above, I also flipped images at random and inverted the sign of the predicted angle to simulate driving in the opposite direction. The flipping of images helped to remove the bias of the track having mostly left turns.

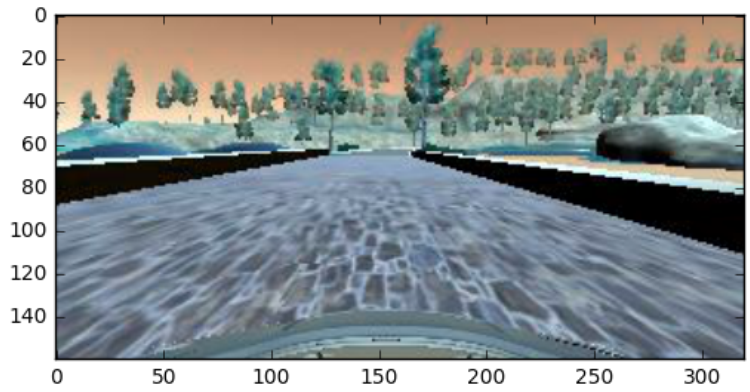


Image before Flipping, steering angle -0.2306556

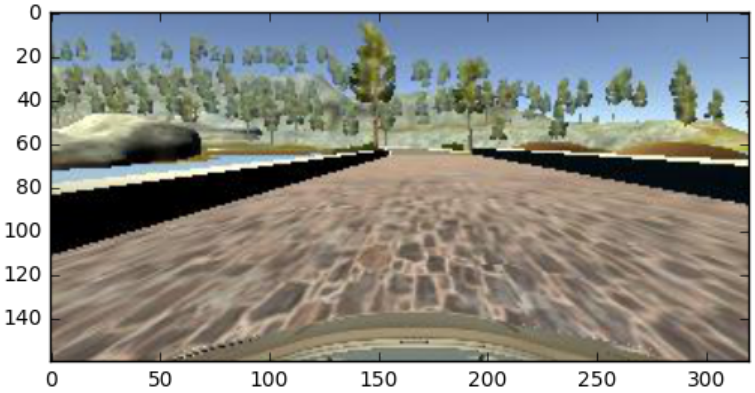
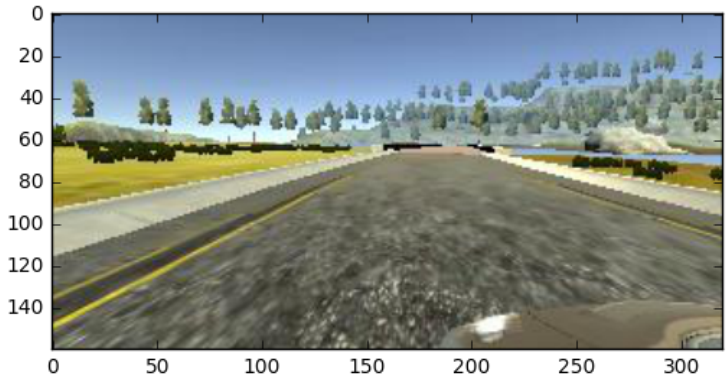


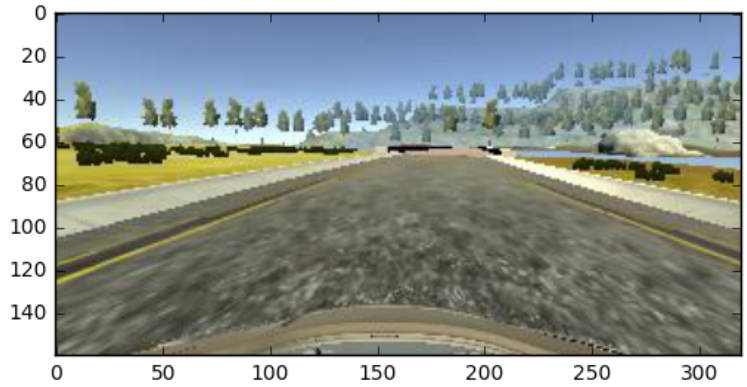
Image After Flipping, steering angle +0.2306556

4.4. Using Left and Right Camera Images

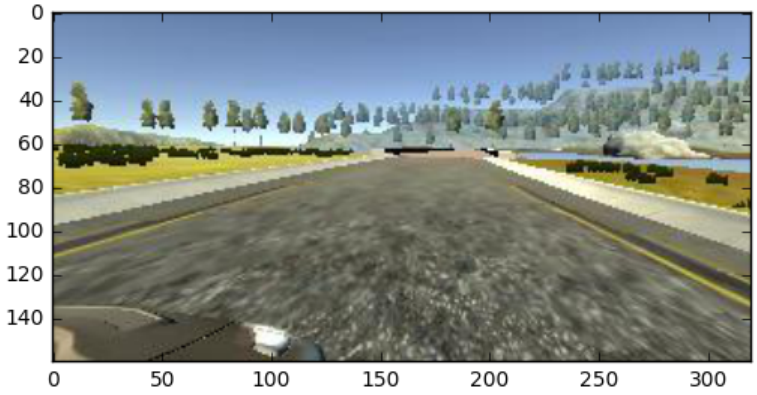
I also used the left and right camera images to simulate the effect of the car wandering off to the side and recovering. The steering angle needed to be adjusted to simulate the center camera images from the left and right camera images. After some trial and error, I adjusted the steering angle by adding 0.25 to the left camera and subtracting 0.25 from the right camera. The main idea was to get the left camera to move right to get to center, and the right camera to move left to get the center camera steering angle.



Left Camera Image, steering angle + 0.25



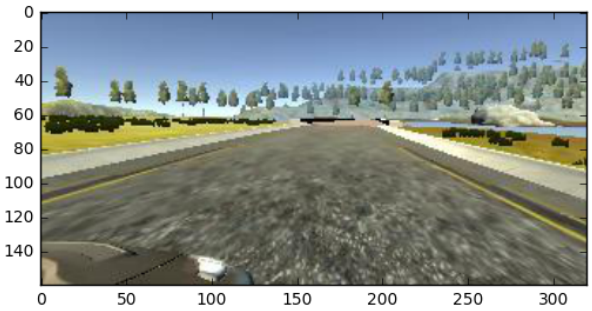
Center Camera Image



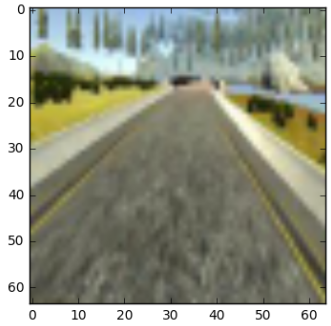
Right Camera Image, steering angle + 0.25

4.5. Resizing the Images

After augmenting the images with techniques mentioned above, I cropped the top 20% of the image to remove the horizon and the bottom 25 pixels to remove the car's hood. Then the image was rescaled to a 64X64 square image. The resizing helps to remove unwanted information without losing fidelity and also helps in reducing the processing time to train the generator. The image before and after resizing looks as follows.



Before Resizing



After Resizing

5. Training Process

The training process used Keras fit generator to reduce memory needs as it is a nice way of feeding data in batches to the model on the fly while generating them. The batch size chosen was 256. I played around with a number of epochs, and finally settled for 5, as a greater number of epochs did not yield any better results. The validation set helped determine if the model was over or under fitting. Adam optimizer was used with a learning rate of 0.0001. I also used Lambda layer in Keras to normalize intensities between -.5 and .5. The Maxpooling and dropout layers in the model address the issues related to overfitting.

6. Final Results

The model worked well for both tracks of the original simulator, although there was a bit of swerving (zig-zag movement) of the car for track 1. By adjusting the speed and steering angle, in drive.py, I could get the car to go a lot straighter.

Surprisingly, the car behaved very well on track 2 (much better than track 1) and stayed on course all the time, although I did not capture any simulator image for this track for training. There were a couple of slopes were the car struggled and started reversing, and so added fail-safe code in drive.py to increase the speed under such conditions. The car stayed mostly in the middle of the road after this modification.

The model did not work on Track 2 of the new simulator due to a totally different terrain and characteristics.

I had recorded the video for posting on YouTube, but the process of recording had a big impact on the simulator performance (too much swerving, although the car did not go off the track) due to the additional

demand on the processing and the memory needs on my machine. Running the simulator w/o recording resulted in much less swerving of the car and the model behaved much better.

I have tested both tracks in all resolutions in the fastest speed mode.

Uploaded YouTube videos.

Track-1 without speed control (*Intermediate model and run*)

https://youtu.be/iE6JpVpRgVo

The car swerves, but stays within the track. I noticed that the swerving is more due to recording. Without recording intervention, as I guess the simulator and the model is processor intensive, the simulation of the car around the track looks more stable.

Track-1 with speed control (*Final Model*)

After modifying the drive.py as per the reviewer comments (modulating the speed with the steering angle), the driving was much smoother albeit slower. Track 1 recording here:

https://youtu.be/4daEsKpvwsc

Track 2, video *(Final model)*

https://youtu.be/DG6a2lVabOA

All these videos were recorded at 640 x 480 resolution and fastest speed settings. Machine: MacBook Pro Model 2015 with 8 GB of memory.

NOTE:

I have tried the simulator for more than an hour without the car going off the track. But running the simulator in the background and doing any foreground work (like editing a spreadsheet or browsing) even for a short while, messes up the simulator/ program and the car goes off track. So, it is essential not to do multiprocessing on a laptop or have a powerful machine in order not to upset the model behavior due to lack of processing power/ memory.

7. Conclusions

The model while working well, is not perfect in any sense and needs a lot more data to simulate and work for other tracks. For example, track 2 of the new simulator did not work from the very first corner as the car could not differentiate two lane lines and multiple roads and I did not train on that track. The major issue I faced in data collection was in controlling the mouse. With a proper gaming controller and plenty of data the model could be trained to work a lot better in most of the conditions and tracks as well.