Behavioral Cloning

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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

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
- writeup_report.md or writeup_report.pdf 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.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 used Nvidia's convolutional neural network (reference 1). The model consists of convolutional layers followed by fully connected layers. The output is only one layer, because the model has to predict a regression type problem and outputs the steering command (code lines 157-174). The input to the model, comes after preprocessing the image (code lines 75-82) in which the images are resized to be the same size as the Nvidia model's input (66,200,3) and also images are cropped and changed from RGB to YUV format. When I used the original image sizes with RGB format, I did not get good results, however I believe, resizing the image and changing the color format helped the model to train well and kept the car on track.

The model includes ELU layers (code lines 160-170) to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer (code line 160).

2. Attempts to reduce overfitting in the model

I implemented many trainings with and without drop out, and my conclusion was that the model performed well when I did not use any drop out layers. When I used dropout, the car was not steering well and getting out of track. I think implementing image augmentation techniques helped the model to train well without overfitting and the need to use drop out layers.

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 173). The loss function used is “mse” because of the regression type problem which I am solving here.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used Udacity data combined with different augmentation techniques. Also, I used all the images produced by center, right and left cameras. For the right and left images, I corrected the steering angle by adding or subtracting the correction factor of about 0.2. I also removed any steering angle less than 0.1 as I believed they really don’t represent the driving behavior.

In addition to Udacity data, I decided to gather my own data in which I drove in both track 1 and track 2. The combination of these data along with the augmentation techniques was also successful in driving the car on track 1. For track 2, the car was able to drive till the half way and I think if I gathered recovery data from track 2 and added to my data, more probably I could drive the car for the full track 2. I will implement this later.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to use convolutional neural network, followed by fully connected layers. Initially I tried using LeNet architecture which I successfully used for traffic sign image classification. Using this model, the car was able to drive for a short run but soon got off the track. The LeNet was used for small size images (32*32), however the image size used in this project was much larger at (66,200). For the next step, with reference to Nvidia’s paper [1], I used their model because they had a success in using this model for real cars. With some adjustments, I was able to achieve the project goal and run the car autonoumsly on track 1.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I was always looking at the possibility of overfitting in the model and if I have to use drop out or any other techniques to overcome overfitting.

My first attempt to decrease the probability of overfitting was to use some augmentation technique which I will describe later. The augmentation techniques are suggested when you have fewer data. Here is a good resource I used.

https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9#.d779iwp28

Using these techniques, I was able to increase my overall data for training and validation. I did not encounter any overfitting in the model and did not use dropout techniques, Infract, I noticed that when I used any dropout layers, my car was not able to follow the track well.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 157-174) consisted of a convolution neural network with the following layers and layer sizes.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 66, 200, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________

flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               115300
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
```

```
Trainable params: 252,219
Non-trainable params: 0
_________________________
```

3. Creation of the Training Set & Training Process

I decided to use the data provided by Udacity first and later I also created my own data by driving on both tracks. I implemented the following pre-processing and data augmentation techniques to increase the ability of the model to generalize. The augmentations were performed on the fly using a custom generator/yield method

Side Cameras

While training the car in the simulator, in addition to center camera images, the images of the right and left cameras also captured. I used theses left and right camera images by assuming that they will impact the steering angle by a value of about 0.25. This value was chosen arbitrary and gives good results. Adding these images will increase the image dataset to simulate the curves.

Image Flipping

I also implemented image flipping function in which images are randomly flipped horizontally. The corresponding steering angle is also changed to be the negative of the original steering angle.

Reducing Zero Angle Images

I tried to eliminate the steering angles of less than 0.1 to reduce the probability of having too much zero angle images and help dataset to balance

Data Size Reduction

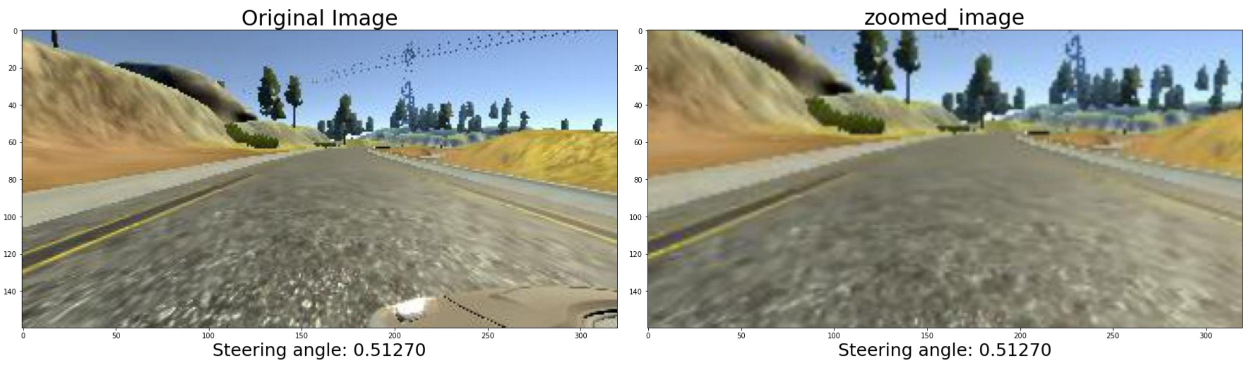
Rather than feeding all the data at once, I used dynamic feeding of images to the model using generator.

Cropping redundant data

Each image consisted of some irrelevant information to training such as background trees, sky and lower part of the car. Cropping those parts of the image not only helped to reduce the image sizes but also help in training the model to give relevant information to the model. The image was cropped 65pixle from the top and 25pixle from the bottom.

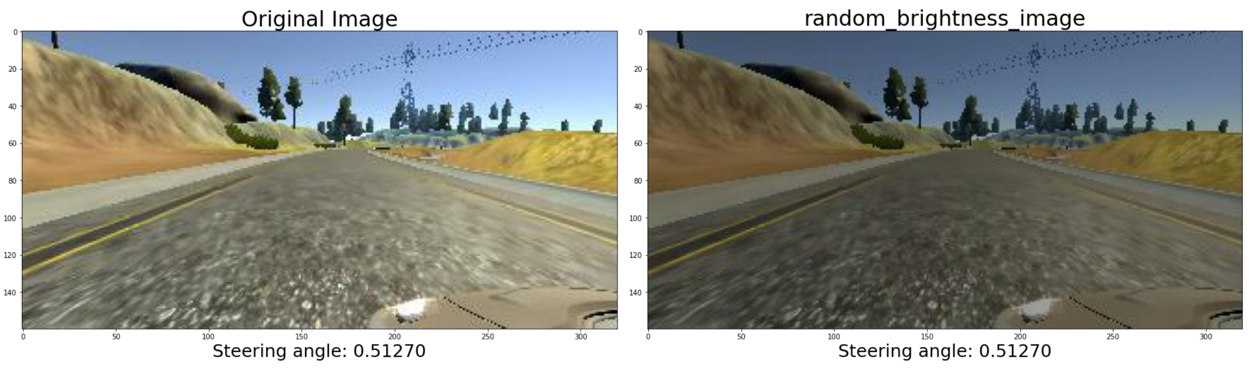
The images below show the augmentation techniques used in the project. These are the minimum techniques I used which gave me the satisfactory results. Adding many other techniques such as random shadow, random panning or shifting might also help.

Zoom: (It will zoom part of the image)

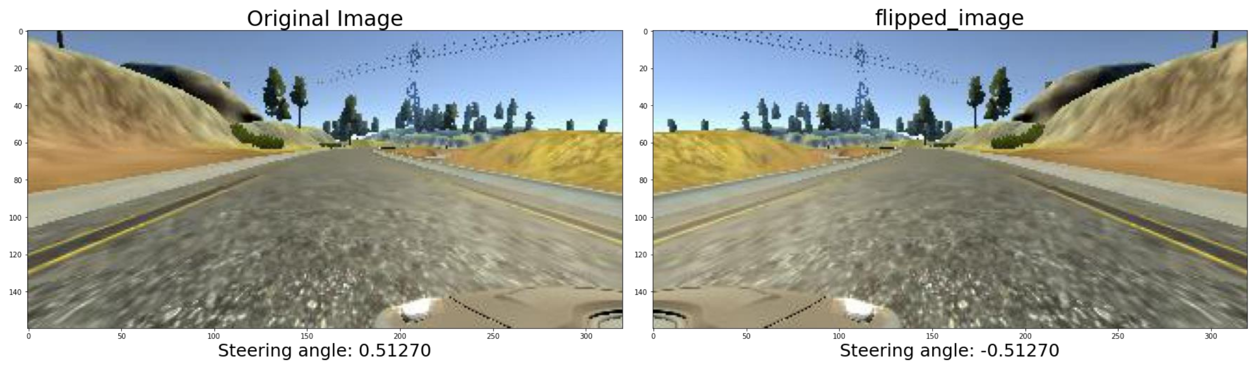


Random brightness: (randomly changing the image brightness)

Since both tracks have different backgrounds with large differences, I thought adding random brightness to images might be beneficial in training and generalizing the model



flip horizantoally: (flipping the images randomly and also adjusting the steering angle by multiplying it by -1)



I finally randomly shuffled the data set by putting the 20% of images to the validation set and used the generator function to feed the images into the model.

The followings are the parameters I used to train the model.

- number of Epochs: 4
- Samples Per Epoch – 500
- Batch Size - 100
- Optimizer - Adam with learning rate 1e-4
- Activations - Elu

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner,B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller,J. Zhang, et al. ***End to end learning for self-driving cars***, arXiv preprint arXiv:1604.07316, 2016