# **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

### 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 has loading and processing data, model architecture and optimizer (line 100 to 112 and line 119) . This also contains comments on how each code functions for its role.

### Model Architecture and Training Strategy

#### 1. An appropriate model architecture has been employed

My model is based on VGG16 model to effectively shorten building and tuning process of the architecture. It has parameters, and its weights are all initialized with 'imagenet'.

Firstly, the model gets an input of the predefined size as row =160, column=320, and channel=3. This is the original size from the center camera.

#### 2. Attempts to reduce overfitting in the model

I did the following steps to avoid overfitting.
I split dataset to train data and validation data by using, so the ratio was 8:2. (model.py line 47) to avoid overfitting. Then I normalized the input data in lamba_1 layer so that the value of each pixel is fixed between -1 to 1(line 106). This is less computationally expensive to process from the input layer.

#### 3. Model parameter tuning

As this model should be designed as a regression model, I used an adam optimizer and the learning rate is included within it. (model.py line 116)

#### 4. Appropriate training data

Firstly, the data should be image data that the car is driving in the middle of the road throughout the track. If a model can perfectly imitate the behavior, the car on simulator will run exactly on the middle of track.

Though due to the mere value of loss functions and so forth, this may be not the case. The car

run by the model will go off from the center of the track, so the model should also combat situations to go back to the center of the track from either side when the car derails from the center.

### Model Architecture and Training Strategy

#### 1. Solution Design Approach

Based on VGG 16 model, it was able to reach less than 0.1 of validation loss with only one epoch. I found this was already high enough as a sign of training, though the problem was that it took about 5 mins to train through the whole dataset. As I add more data of edge cases for training data, I needed to train the model again with longer training time. After every iteration of this data-collecting and training, I ran the car on the simulator and I ended up having the satisfying result after 10 trials.

I tried to use some different models such as InceptionV3 and MobileNetV2 to shorten this iterative time, though this wasn't successfully executed due to the version of tensorFlow-gpu. (As far as I looked up, the version should be more than 1.5.0. But it was 1.3.0, as I searched it by 'pip – list ' command on bash).

If this went healthy, the parameters used in the network would have decreased as both of models have less parameters. Then the computation process would have taken much shorter, thus it would have allowed me to iterate this process more.

This time, I only fed the neural network with the car's steering angle and wanted the network to execute a steering angle depending on input camera images on vehicle in the simulator. So the output layer is a dense layer of single output.

After feeding dataset of edge cases, the car could drive around the track autonomously without causing any troubles.

#### 2. Final Model Architecture

The final model architecture (model.py lines 18-24) is as follows.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 160, 320, 3)       0
_________________________________________________________________
cropping2d_1 (Cropping2D)    (None, 90, 320, 3)        0
_________________________________________________________________
lambda_1 (Lambda)            (None, 160, 160, 3)       0
_________________________________________________________________
lambda_2 (Lambda)            (None, 160, 160, 3)       0
_________________________________________________________________
vgg16 (Model)                multiple                  14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 12801
=================================================================
Total params: 14,727,489
Trainable params: 14,727,489
Non-trainable params: 0
```

After taking an input, I cropped the image from 160x320x3 to 90x320x3 to speed up computation while focusing on necessary images in the record : the sky and tress in the top realm in the picture does not affect on driving.

I changed the input size to VGG16 model to 160x160x3, to reduce the number of the parameters (thus speed up the computation). This is a trade-off of decreasing the model accuracy as it has less parameters, but this size was appropriate to successfully execute a desired result.

As Vgg16 by design ends with a fully connected layer of 1000 different outputs, I removed it, flattened it and output 1 result for the reason above.

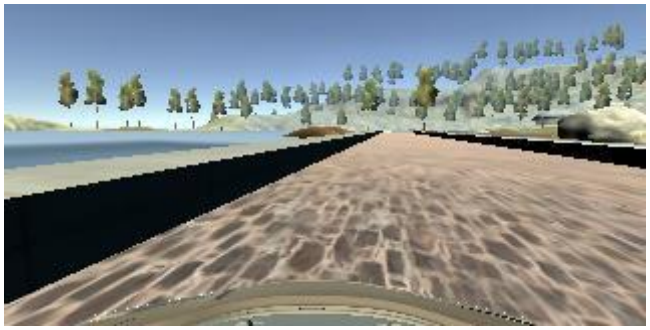
#### 3. Creation of the Training Set & Training Process

Firstly, I chose data that was originally stored in this path 'opt/carnd_p3/data' and ran the model.py. The car was able to reach to the end of the bride in the first track without having any troubles, though it ended up crashing to the right side of the bridge.

Then I took additional data to train the car for edge cases, typically some scenarios to recover from off the road to the center. I ran the car clock-wise on the track in training mode on the simulator to learn curving of right from left. Then I ran the car counter clock-wise in parts that were either ; the car needs to turn left on the road with particularly intense curvature; the car on the bridge trying to go to the center from either edge of the road.

Below is an example picture where the car was running clock-wise.

Here is another example image, where the car was about to hit the edge or in the side of the road. After this I oriented to the center.



This is another example of when going to the center from the right side of the road.

The end result is that the data has about 270MB, and it has 17578 images as center images. After splitting the data as I mentioned above, I fed these images to train the network and validated it.