# Behavioural Cloning Write-Up

## Introduction

As part of this project I utilized the concept of behavioural cloning to make a car drive itself around a closed course. In order to do this, I first trained the car to drive around the course and using my model, trained it to go over the entire round.

Data Collection

## Strategies

In order to collect data for this project, I used the following strategy using only data from the first track.

1. Drive 3 clean laps at roughly the centre of the lane
2. Drive 2 clean laps in reverse around the course
3. Record instances of coming avoiding the boundaries in several different instances. I avoided the following boundaries
   1. Yellow lines on either side
   2. Road Boundaries on either side
   3. Dirt on the side of the track
   4. Kerbs (The red and white things)
   5. The boundaries on the sides of the bridge
   6. Confluence points of different types of boundaries as described above.

Pre - Processing Data

## Possibilities and Choices

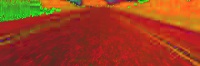
The input image (320x160) offers a lot of information. However, there are several parts of the image that offer little to no information to help training. It’s important to eliminate these features so that they do not interfere while training the model. I have highlighted some of these features below



As you might notice, there’s a ton of data we don’t need. There are several ways, we could eliminate the unnecessary data. I’ll go over a couple of things I thought of before I tell you what I did.

1. Colour Filters
   1. We could filter out just the colours for the road aka grays and yellows. This is easy to do and will work great in the case of this image.
   2. However, for images with more colourful roads like the section with bricks, this will fail miserably, hence I avoided this
2. Lane Detection [NOT IMPLEMENTED – A possibility for later]
   1. Similar to what we did in project 1, I would detect the boundaries of the road and enhance those lines in the image by overlaying a strongly coloured line on top of the boundaries of the road
   2. Additionally, I would mask out other unimportant parts of the road that are unimportant.
   3. This sounds like a really cool thing to do, and I will try it in later submissions, but for this one, I will go with just what I wrote below.
3. Cropping the image
   1. This is the simple method I used to remove unnecessary data
   2. Cropped the image using the following criteria
      1. In the Y direction, I chose all pixels between rows 55-136
      2. In the X direction, I chose all pixels between columns 0-320 (effectively cut nothing)
4. Switching Colour Models
   1. I switched the image from RGB model to the HSV model as I wanted to train the model to understand an image by separating the luminance information from the chroma information
   2. I also found that this model works much better than simple RGB which with all respect, performed quite poorly.
5. Scaling Image
   1. I scaled the image to a size of (200x66) to try and make it similar to the Nvidia model discussed in the next section

At the end of my transformations, this is what the output image looked like.

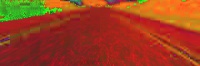


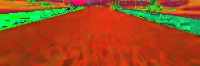
Sample Training Data

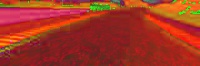
## Examples Before Pre-Processing



## Examples after Pre-Processing (Same Order)







Model Selection

## Inspiration and Custom Model

There are several models out there that are specialized for self driving cars. Among those is the model from Nvidia that is described in [this](https://arxiv.org/pdf/1604.07316.pdf) paper. To be honest, I initially tried this network as I believed that it’s tried and tested and should work pretty well. It would have been great if this was available with its weights as well as that would enable me to use transfer learning in order to solve the problem faster. However, the weights are proprietary and this model takes a very large amount of memory which caused my instance to crash several times.

I persevered and didn’t lose faith. Using this as inspiration, I devised my own model as follows

1. **Lambda Layer for normalization**
2. **Convolutional Layer**
   1. Depth of 8
   2. 5x5 Kernel Size
   3. Relu Activation
3. **Convolutional Layer**
   1. Depth of 16
   2. 4x4 Kernel Size
   3. Relu Activation
4. **Convolutional Layer**
   1. Depth of 32
   2. 3x3 Kernel Size
   3. Relu Activation
5. **Fully Connected Flatten Layer**
   1. Dropout with Keep Probability of 0.3
   2. Relu Activation
6. **Fully Connected Layer**
   1. 128 Neurons
   2. Dropout with Keep Probability of 0.3
   3. Relu Activation
7. **Fully Connected Layer**
   1. 10 Neurons
   2. Dropout with Keep Probability of 0.3
   3. Relu Activation
8. **Output Layer**
   1. 1 Neuron

Training

## Generator

In this case, I had to make use of a generator to train my images since I faced several memory constraints. I used a generator based on the sample code provided by Udacity that also conducted my pre-processing steps for me

## Training Methods and Parameters

Loss Function : Mean Squared Error

Optimizer : Adam

Epochs : 10

Batch Size : 8 (Small due to memory constraints)

Testing

## Simulator

I used the Beta Simulator to test my results. It should technically work with the stable version as well but I did not perform exhaustive tests there.

## Modifications

1. Reduced the throttle from 0.2 to 0.05 in drive.py to reduce the speed to the car as higher speeds were not working properly with my slow machine
2. Also performed the pre – processing steps highlighted earlier on the images from the simulator before sending them to the model to be trained.

## Results

1. Track One : Hurray! My car completed this circuit by staying within the boundaries of the road for one complete lap.
2. Track Two : The model did not generalize well from the training data which was solely track one.

Improvements

## Data Collection

Certainly, I can collect data from both the tracks and try to make it run on the second track as well. This would be pretty cool and I am sure I can do it in a later submission or do something in my free time.

In order to get better quality data, I could also use a PS4 controller to generate smoother angles during the training phase. I was able to get it working but my test data I used did not use it because I had already collected decent test data by then. However, this could probably make it perform 10-15% better.

## Pre - Processing

There’s certainly a lot of wizardry that can be done here, some things which I highlighted above. I would definitely go and visit these things in later attempts.

## Model

If only I had more memory! Certainly, I could try this out on one of the large g8 clusters as well and try to see if I can run Nvidia’s network on it to see how it performs different to my model. However, in computer science a lot of times less is more. Lesser memory consumption and a shallower, slimmer model means I can run my test cases really fast.