Behavioural Cloning Report

1 Overview

I used an End-to-End Deep Learning approach to learn the steering angle prediction task.

The main steps were to:

- Using the simulator to collect data of good quality.
- Use an End-to-End model to train the network.
- Improve the model to generalise on a difficult track.

1.1 Dataset Creation

The Car in the Simulator takes 3 images per timestep.

- Centre Image
- Left Image
- Right Image

Figure 1: Centre Image

Figure 2: Left Image

Figure 3: Right Image

For data collection, I used only the "Simple Track".

Figure 4: Simple Track - Track 1

Figure 5: Complex Track - Track 2

During driving the car on simulator, the output is a single steering value for all the 3 images, so I used a steering correction for the left and right image.

From the perspective of the left camera, the steering angle would be less than the steering angle from the center camera. From the right camera's perspective, the steering angle would be larger than the angle from the center camera.

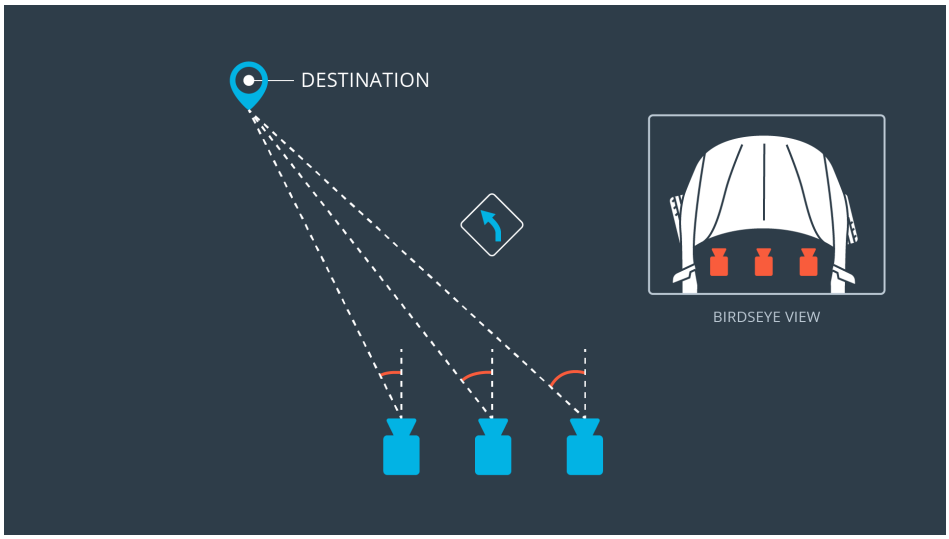

Figure 6: Angle between destination and each camera

I used a correction factor of +0.2 and -0.2 for the left and right images respectively.

This gave me data corresponding to drifting off of the vehicle, i.e. if the vehicle drifts off then how it should come back. A very few adversarial examples were also collected to give as input.

Since Track 1 is counter clockwise, so I used data augmentation and flipped each image along with the steering angle values to make the pipeline robust.

Figure 7: Original Image

Figure 8: Flipped Image

1.2 Pipeline

Preprocessing

Different models were trained based on different preprocessing methods. Mainly there were 3 case:

- Using RGB image
- Using HLS image
- Using only L and S Channels of HLS image

Model Architecture

The model consists of the following layers:

- Cropping2D
- Normalisation
- Conv2D - 5X5 filters, num-filters = 24, strides = (2,2), act = 'relu'
- Conv2D - 5X5 filters, num-filters = 36, strides = (2,2), act = 'relu'
- Conv2D - 5X5 filters, num-filters = 48, strides = (2,2), act = 'relu'
- Conv2D - 3X3 filters, num-filters = 64, act = 'relu'
- Conv2D - 3X3 filters, num-filters = 64, act = 'relu'
- Flatten
- Dense - 100 units, act = 'relu'
- Dense - 50 units, act = 'relu'
- Dense - 10 units
- Dense - Final output (1 unit)

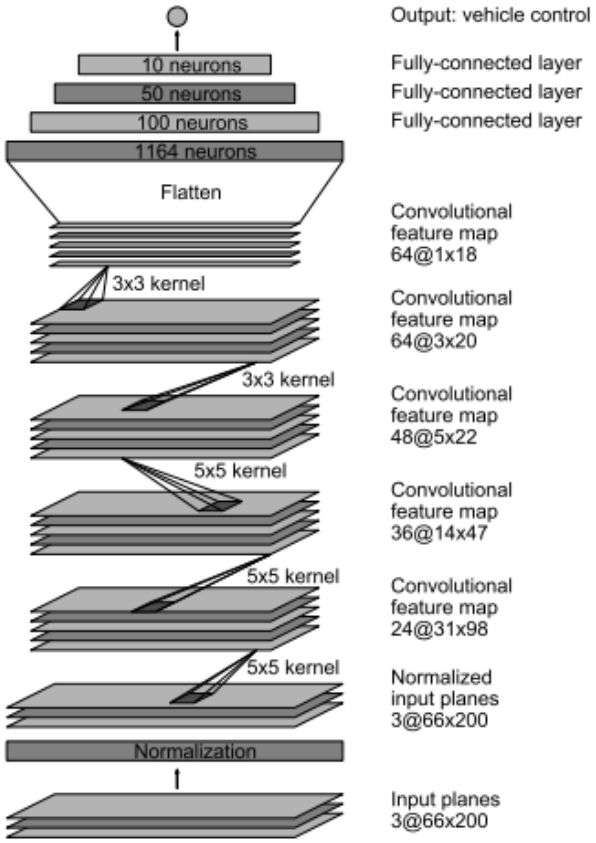

Figure 9: Model Architecture - 250 thousand parameters and 27 million connections

Training

The dataset was divided into 0.8 : 0.2 ratio of train and validation sets. **MSE** loss was used to train the network with **Adam** as optimizer.

1.3 Results

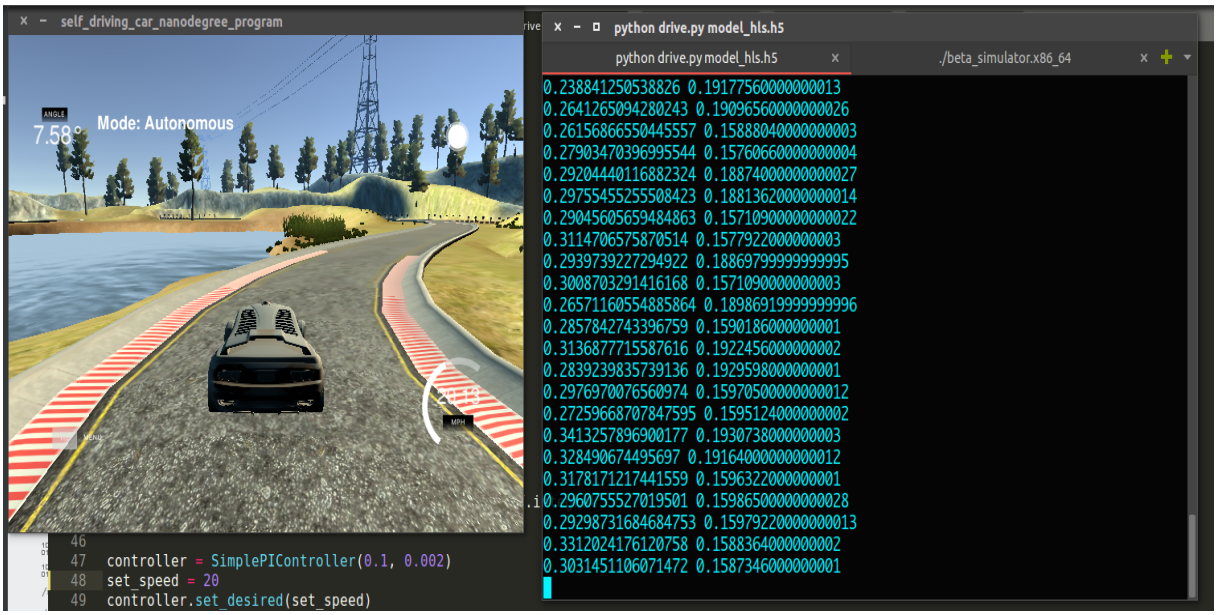

Figure 10: Track 1 - Autonomous Mode

Figure 11: Track 2 - Autonomous Mode

The most remarkable result of this project is its Generalisation. As mentioned, for collecting the training data only "Simple Track (Track 1)" was used, whereas my model performs outstanding on "The Complex Track(Track 2)" too. The RGB model fails to generalise whereas both the "HLS" and "LS" models generalise very well on the tracks.

This generalisation shows that the model hasn't remembered the track and has learned features which helps it identify edges and other parameters necessary.

Autonomy Metric

We estimate what percentage of the time the network could drive the car (autonomy). The metric is determined by counting simulated human interventions. These interventions occur when the simulated vehicle departs from the center line by more than one meter. We assume that in real life an actual intervention would require a total of six seconds: this is the time required for a human to retake control of the vehicle, re-center it, and then restart the self-steering mode. We calculate the percentage autonomy by counting the number of interventions, multiplying by 6 seconds, dividing by the elapsed time of the simulated test, and then subtracting the result from 1:

$$\text{autonomy} = \left(1 - \frac{(\text{number of interventions}) \cdot 6 \text{ seconds}}{\text{elapsed time [seconds]}}\right) \cdot 100$$

Figure 12: Autonomy Metric

The autonomy results are:

- For RGB model trained on Track 1
  - Track 1 - 100 % (speed limit = 30mph)
  - Track 2 - 28.75 % (Total Simulated time = 160sec, Interventions = 19 , speed limit = 12mph)
- For HLS model trained on Track 1
  - Track 1 - 100 % (speed limit = 30mph)
  - Track 2 - 83.33 % (Total Simulated time = 180sec, Interventions = 5 , speed limit = 12mph)

As one can see, the HLS model gives far greater generalisation results as compared to the RGB one.

1.4 Future Work

- Running the model in a more complex environment.
- Using various weather conditions, check for robustness and improve the model.
- Incorporating Traffic in the simulation and Traffic rules following.