CarND Behavioural Cloning

By James Marshall

Data Collection and Augmentation

For this project, the data that had to be collected was incredibly important, as the model was training to drive exactly as the data showed it to. This means that any poor driving behaviours such as late or early turning, swerving, etc, would be copied by the model and the car would fall off of the track. I think I recorded new data at least 10-15 times, as for a long time, I could not allow the car to stay on the track, even though both the training and validation loss both seemed to be low.

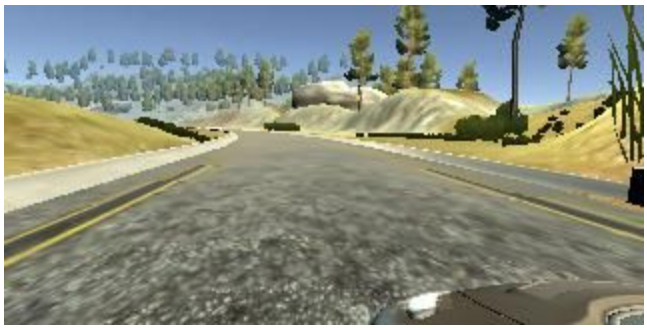
My final set of data for the model, which allowed the car to drive around the track endlessly, without coming off at any point, consisted of two laps the correct way around the track, sticking to the middle of the road; two laps in reverse to help with left turn bias and to better generalise the model, and also driving through the first curve after the bridge twice more, as this was the one place my model seemed to be failing.

Once I had my data collected, I had to augment it. As a large portion of the track consisted of little to no turning, there was a large proportion of extremely low steering values. To overcome this problem, I both removed data where the steering angle was 0, and secondly, made a histogram of steering values, divided into 25 bins, and any bins that had more than 1.3 times the mean amount of values per bin, were brought down to 1.3 times the mean. This gave a much more normal distribution of values, not allowing the model to get away with driving straight all of the time and maintain a god accuracy.

As all of the data was taken from the centre of the lane, I had to make sure my model could correct itself, if it ever came off centre. I did this by taking the images from the side of the car also and applying a correction value to the steering angle, ± 0.25 as shown below:

Centre Image

Left Image



Right Image

Once I had all of my data collected and ready to go (approximately 10,000 samples), I just had to do some preprocessing of the images, before they were fed into the model.

As I followed the nVidia model for a long time with this project, I followed a lot of its tips, even after I changed the model slightly. This is why every image was changed into the YUV colorspace and resized to 200x66. Before the images were resized however, the top and bottom of each image off, as to eliminate the redundant data in each image (the sky and bonnet of the car).

I also followed the advice of Jeremy Shannon and his sample implementation and decided to augment the training images as well, to help further generalise the model. For each training image, the brightness was randomly adjusted, a rectangular portion of the image was shadowed and the horizon of the image was also warped, which helps for roads that contained an incline (track 2).

Model Architecture

Initially, I was using the nVidia architecture, which seemed to have low validation and training losses (approx 0.01), however, I could never get the car to stay on the track, the entire way around, regardless of how I collected the data. This lead me to searching for other, similar architectures, and I was able to find a model created by Moritz Cremer, another student who used a very similar, but tweaked model to that of nVidia.

The model consists of four convolution layers, which have increasing depth (32, 32, 64, 64 in order) and decreasing kernel sizes and step sizes (7, 7, 5, 3 and 2, 2, 1, 1 respectively). Each of these layers were followed by an elu activation function and had two dropout layer using a dropout probability of 0.3 after the second and fourth convolutional layers.

After these convolutions, the model is flattened and sent through four fully connected layers. Each of these layers decreased in size and were followed by elu activation functions and dropout layers, again with 0.3 dropout probabilities, after the first two fully connected layers. The last layer outputted one value, the predicted steering value.

This model had much lower training and validation loses than the nVidia model, reaching losses as low as 0.0007, however, as I discuss later, this was not necessarily the best model.

Model Training

Firstly, the data had to be split into training and validation sets, to show us if the model was over or underfitting. I made an 80/20 split of the data, which is widely considered a good ratio for splitting data.

The model was trained using mean squared error and the learning rate parameters were controlled by the Adam optimiser, with an initial learning rate of 0.0001. The images were fed into the model using a generator in an effort to save memory. The model ran through the entire training set for 15 epochs, with the weights being saved after every epoch as a checkpoint, and was set to stop early if the validation loss didnt get a new minimum after 3 epochs in a row. What I found most interesting, was that when I found a combination of model and data that allowed to car to complete the track, the best model was not the model with the lowest validation loss, which proved the checkpoints to be vital as I didn't lose the best model by overtraining and overfitting.

Overall I would like to collect more data for this model from track 2 and test it there, as I have already set up image preprocessing for the track, but have not had the time to try and implement it yet. This may give me insight to problems with my model and I am excited to solve those also.