Behavioural Cloning

Seymur Dadashov

October 27, 2020

Abstract

The purpose of the following project is to be able to create a and train a Convolutional Neural Network to successfully read in images from cameras mounted on a vehicle within a simulated track environment in order to produce a model that can mimic the behaviour of a human driver that has driven the same identical track. The method is known as behavioural cloning and is a great way of bringing human intuition and understanding into autonomous systems as the data being fed into the model is coming directly from a human source.

1 Method of Approach

The initial thought of attempting to solve the following problem was to simply create a set of data of myself driving the car around the simulated track and then "plug-and-play" that data into the model. Unfortunately my thought process proved to be very naive and it quickly became the case that this was not such a simple problem.

During the initial few tries I had generated a set of data of myself driving the track for two laps and then driving the same track in the reverse direction for another two laps. I believed that this would provide sufficient diversity in terms of the steering angles being used for training and the images that the model was seeing. Naturally I could also flip the image and steering angle to produce the same affect but I felt that the data size was going to be too small. Additionally driving the track in reverse produces some slightly different images as I cannot perfectly drive it in reverse as I had done forwards.

The training images were barely pre-processed with only some Gaussian blurring applied and fed through the following recommended CNN architecture below.

Upon completing training it was very obvious I had a lot of work left to do just to make it around a portion of the track.

2 Approach Revisited

2.1 Data Pre-processing

One of the first things I had began to look at which I initially skipped over was the distribution of the steering angles and how many samples were found across general grouping of the steering angles. From the beginning it was very obvious the entire track had a bias to turn left and I had expected a large majority of the distribution to be skewed to the left steering angles. In addition to the distribution it was to be noted that data was incoming from three different cameras; however the angles that were assigned to those cameras were entirely the same. Visually this means each side camera direction of focus or in other words the angle at which the side cameras are looking at is parallel to the center camera but never converging to the point of interest the center camera is looking. Figure 2 clearly demonstrates the point I mistakenly looked over.

Upon creating a steering angle offset for the two side cameras such that they better represent the behaviour in Figure 2, the steering angle distribution had to be more balanced out in order to get rid of the skew. Figure 3 displays how the raw data was distributed. Figure 4 shows the outcome of the data processing that I had done and then used the following distribution to train the model.

To begin with the distribution as seen in Figure 3 is very clearly multi-modal defined by the large peaks that correspond to left, middle and right steering angles. The reason these angles are so dominant is because

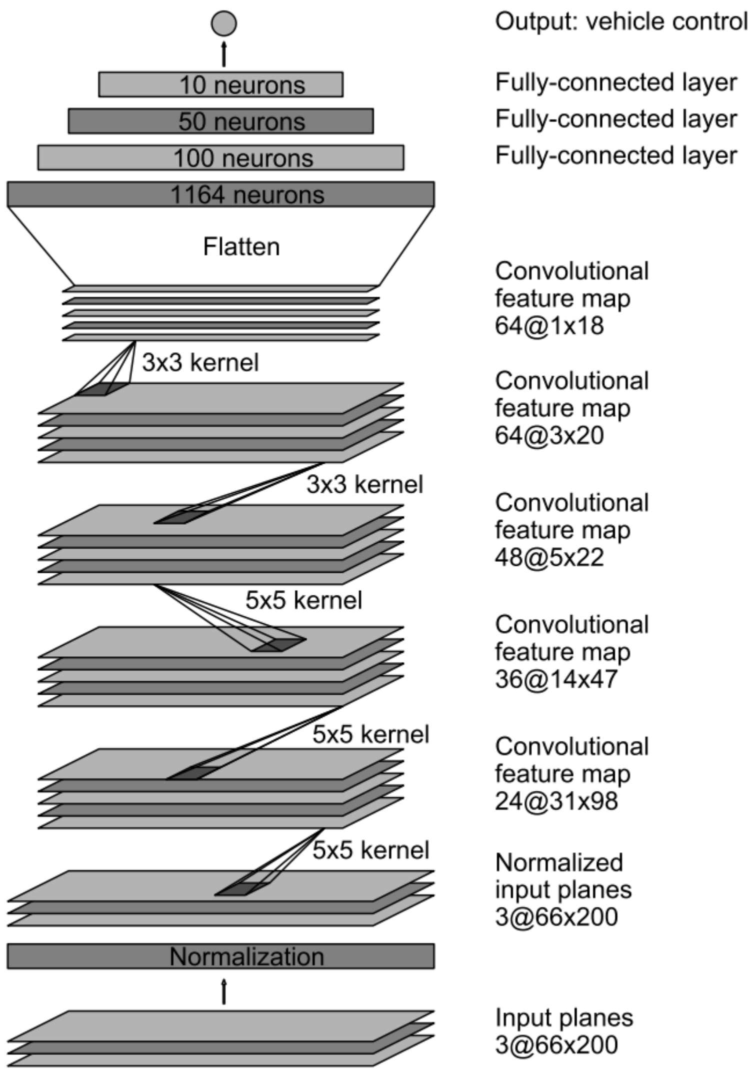

Figure 1: Nvidia Convolutional Neural Network Architecture

the track tends to maintain roughly the same type of turns throughout. For the purposes of training the model, having this many samples of the common turns would prove pointless as at some point the model will "just get it" for these specific images + steering angle combinations. I used an average value across all bins to determine the cropping point seen by the horizontal line and further reduced the peaks by a fraction of the already cropped data to give angels with small representation a better chance to be trained on. It should be noted that $steering\_angles > |0.4|$ were cropped out as they were simply demonstrating my poor driving and other small jitters during the initial recording of the data.

What was truly a challenge to work with is getting the exact correction factor correct for the left and right steering cameras as this directly caused a shift in the distribution of the data seen in the histograms. This should make intuitive sense since I am directly modifying the values of steering angles and thus how the angles get binned. After several hours of experimenting to get the model where I wanted it to be, I had settled on the value of $steering\_correction = 0.315$ as seen in the `function.py` file.

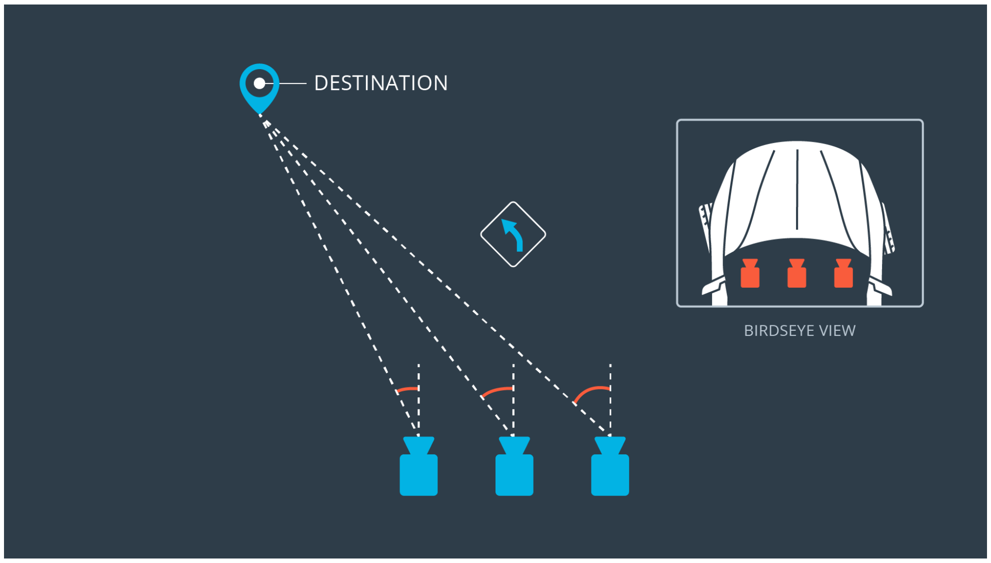

Figure 2: Multi-Camera configuration and steering angle observation

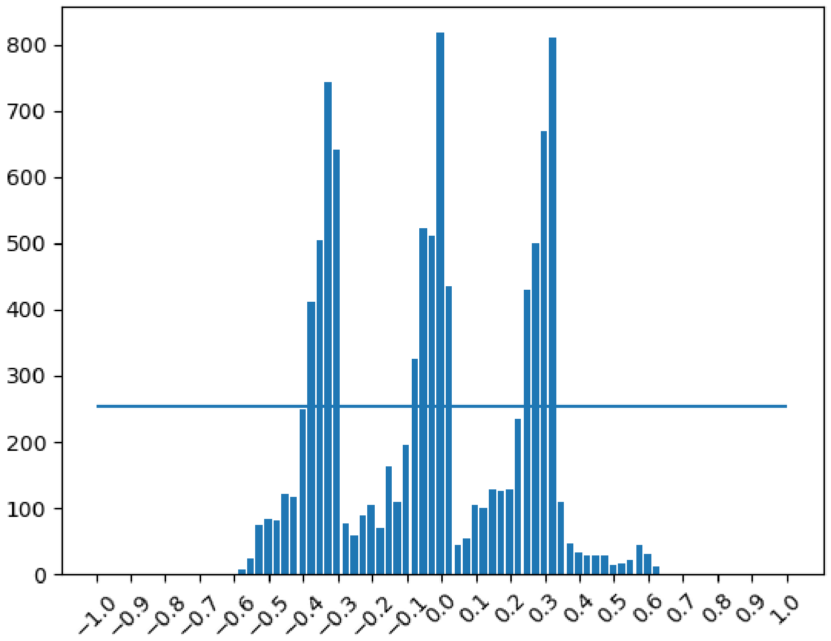

Figure 3: Distribution of steering angles prior to steering angle correction and filtering.

2.2 Image Pre-processing

Despite being the least impactful when compared to the large difference that cleaning the initial data created, image modifications such as cropping and producing some distorted images for the model to train on did prove helpful in the end. I did not utilize anything particularly special expect for the recommended color space changes by the Nvidia paper prior to feeding the images into the model and some cropping of the images to get rid of the hood of the car and excess environmental surrounding. I also experimented with

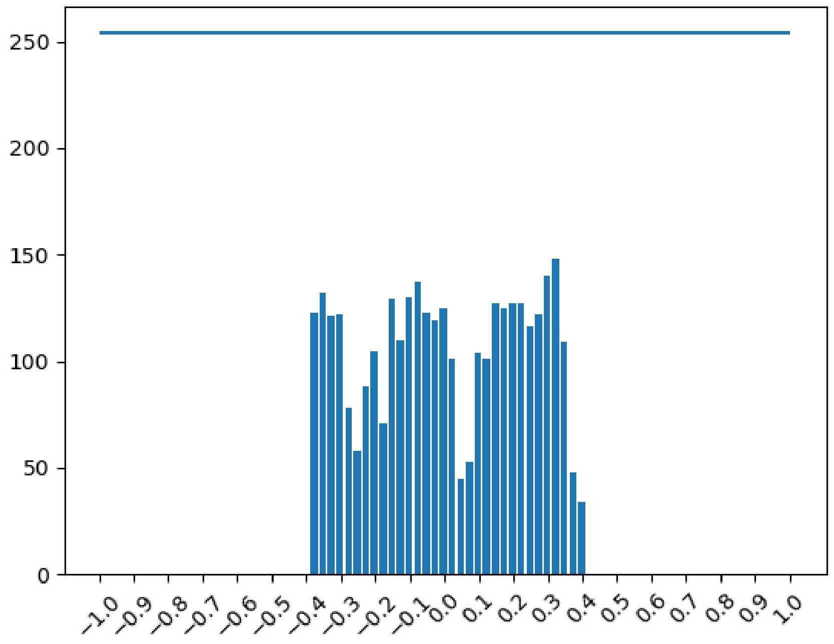

Figure 4: Distribution of steering angles after to steering angle correction and filtering.

altering the image brightness and rotating the images around in space to see if I can make it more robust, but realistically the most impactful would be only changing the brightness as there should be no point where the road magically begins to tilt so that my translation/rotational distortion prove to be useful.

3 Conclusion

In the end, the most valuable take away from this project aside from gaining practical experience working with simulated data and a test vehicle, was truly understanding how to work with incoming raw data and the procedures required to clean the data before feeding it into a CNN architecture. A lot of machine learning and deep learning appears to be a black box when it comes to creating these models, but realistically one can build a very good intuitive understanding of how the data he/she plug into the system will turn out in the end. The same applies to the architecture and the structure of the CNN as one can peek into the layers to see what is occurring although at times it may be difficult to understand, it should not be treated as a black box especially when we have a desired goal in mind. The machine should not have to interpret what we are thinking and guess the outcomes of that black box we made it into.