Model Architecture and Training Strategy

My model consists of a convolution neural network with 3x3 and 2x2 filter sizes and depths between 32 and 256.

The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in between the fully connected layers order to reduce overfitting.

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. I used the udacity dataset for training and did a 10% split for validation dataset.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. Data from all three cameras, canter, left and right has been used with a select_img function which randomly uses one of the images at one time. This gives a strategy to not rely on one camera and also capture more features from the side cameras.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to begin with an already built architecture, I started with LeNet, found it to be not a good fit for this task as the car went off the track. Then I switched onto Nvidia's network, which took a lot of time to train, so I tweaked it and went onto train on images of size 100x100.

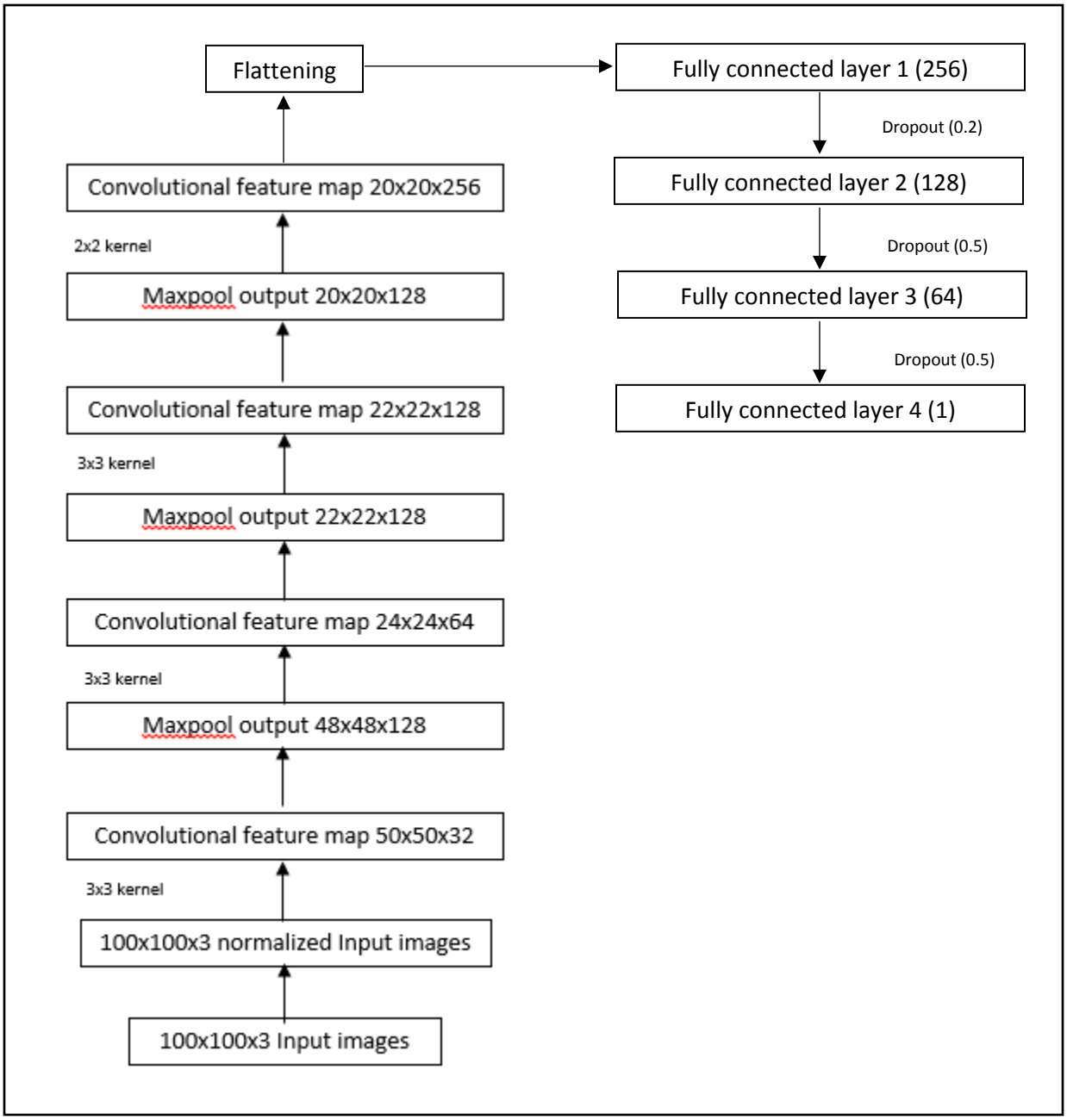
The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track and to improve the

driving behavior in these cases, I introduced a flip_images function which randomly flipped images and the steering value which prevented the left turn bias of the network.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

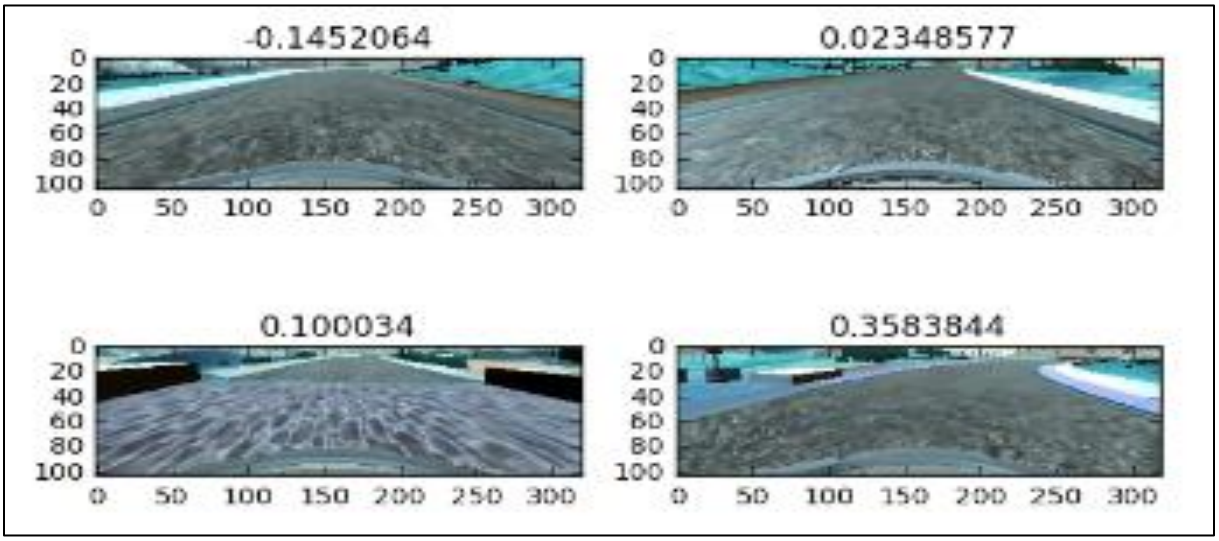
2. Final Model Architecture

The final model architecture consisted of a convolution neural network with the following layers and layer sizes.

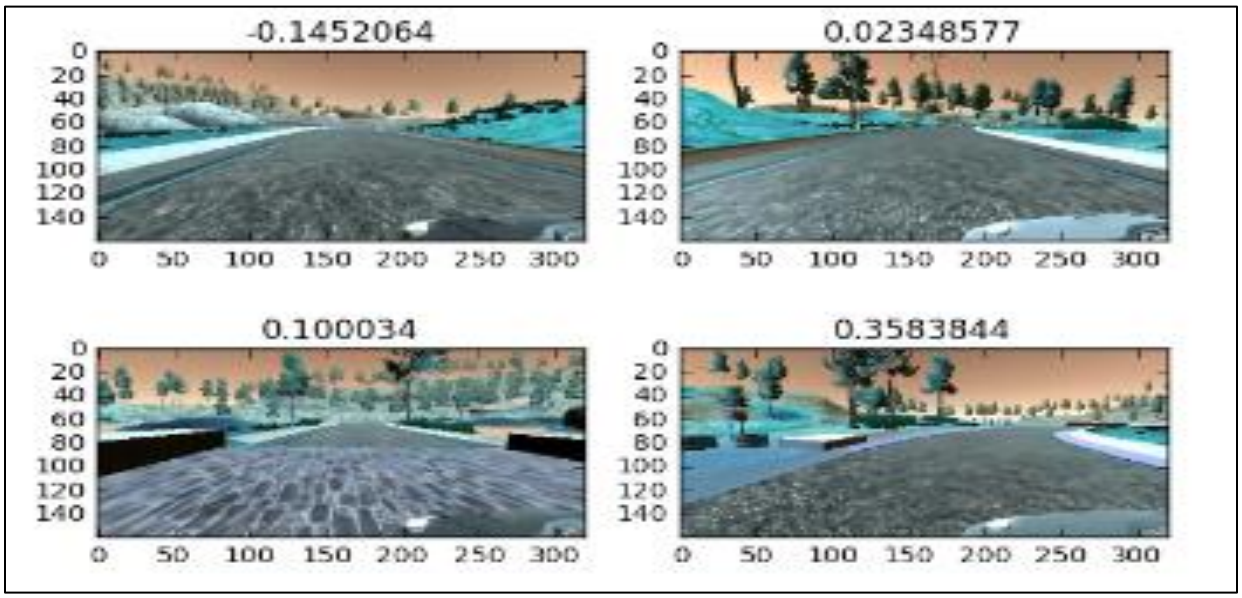


3. Creation of the Training Set & Training Process

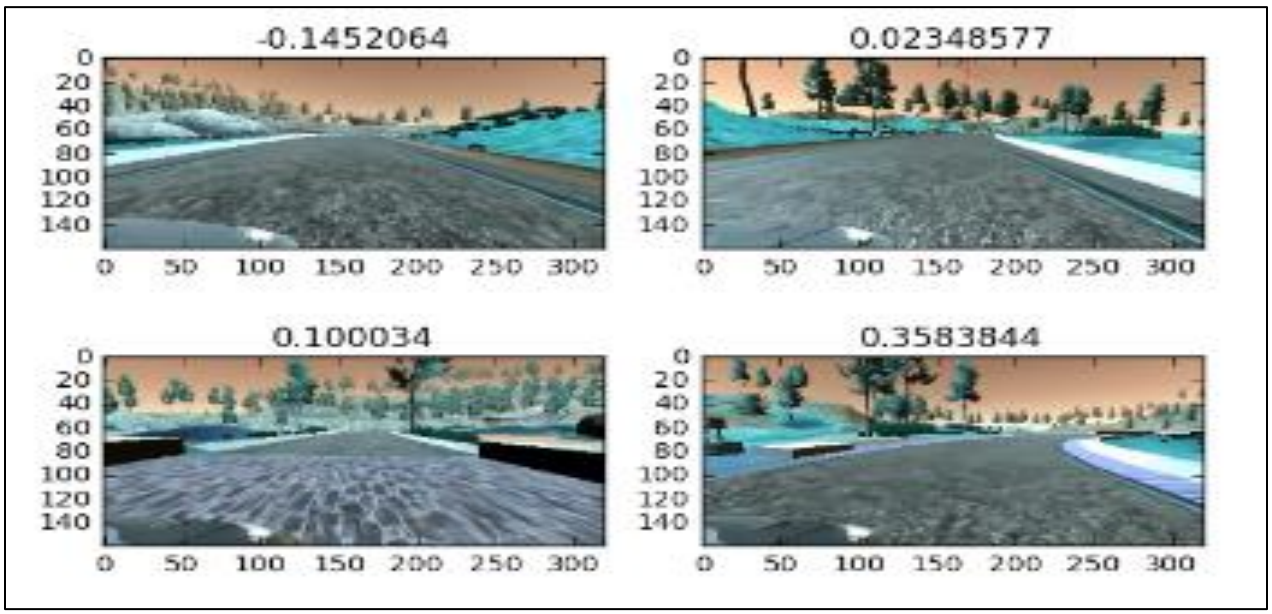
I started by collecting my own data and collected about 4000 images from each camera but it turned out to be not of a big use as the results obtained from it were not comparable to those obtained from udacity dataset. Controlling the car using the keyboard was a big pain and so I decided to take full use of the udacity dataset and randomly input images from center, left and right cameras to capture multiple features. Here are the examples of image from center camera:



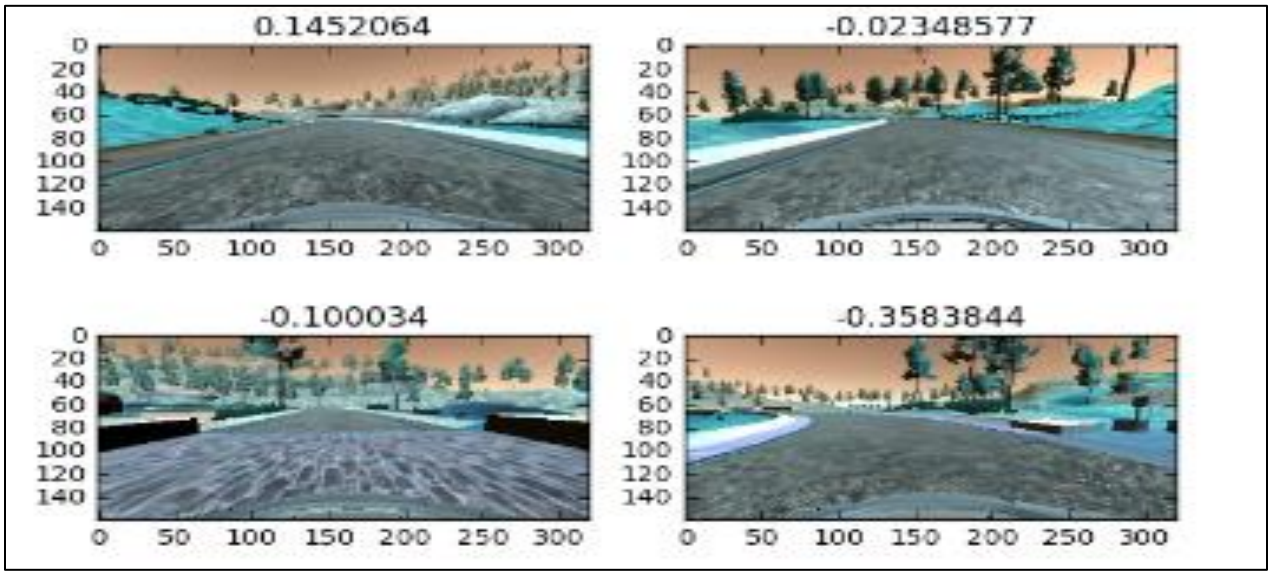
Left Camera Image examples:



Right Camera Image examples:

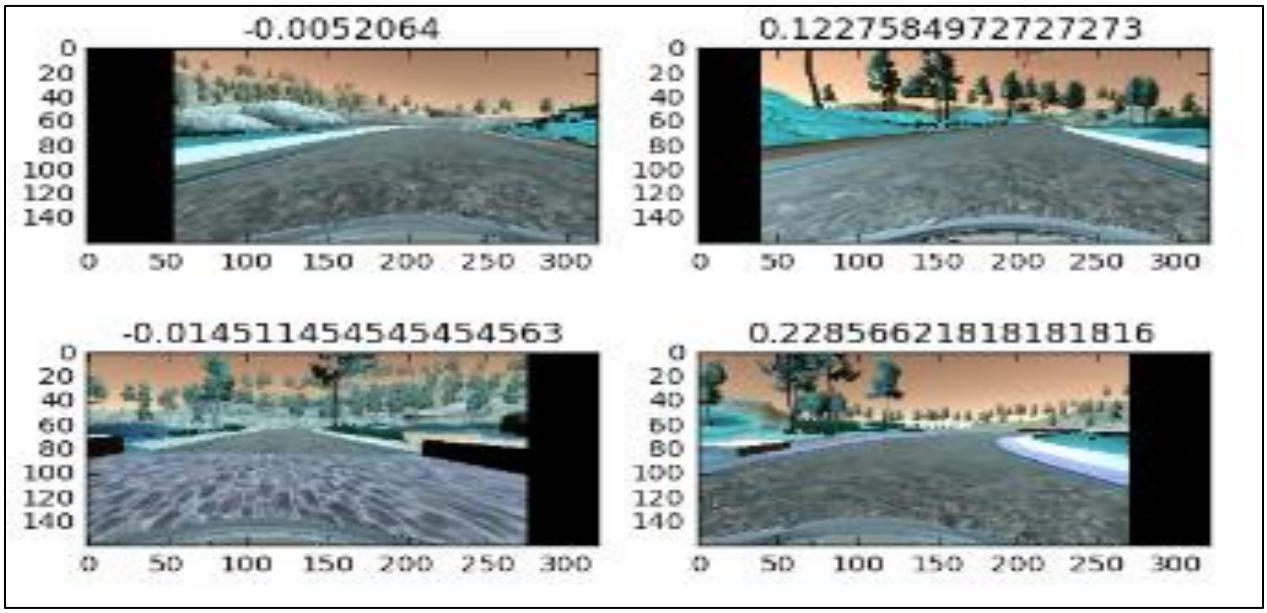


To augment the data sat, I also flipped images and angles to prevent left turn bias of the network as the track 1 had a lot of left turns. For example, here are the above images after being flipped:



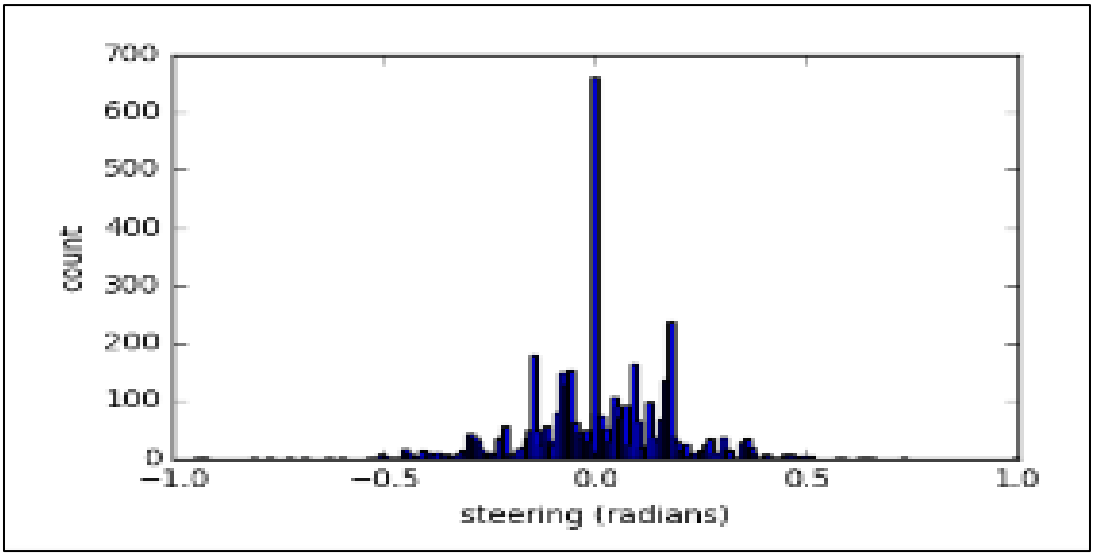
I decided not to train the network on track 2 because of lack of time and my engagements in my Masters degree right now. After the collection process, I had 20480

number of data points in total from 3 cameras. I then preprocessed this data by resizing the images to 100x100, cropping out the unnecessary regions and removed the upper 30% part of the image because it mostly had sky and didn't add to the training. I also added the shift_img function to shift the images horizontally in a random manner so that the network could get a feel of what to do if it's in a scenario very near to the edge of the road which is what shifting simulates. This technique has been used in the Nvidia paper too. Following are the above images after applying shift_img function:

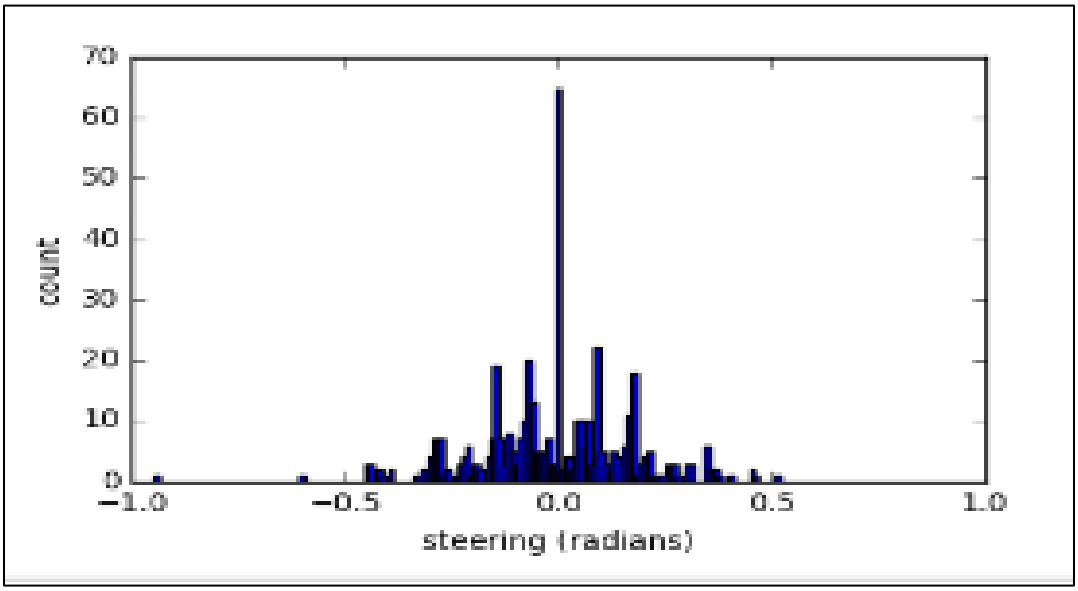


I finally randomly shuffled the data set and put 10% of the data into a validation set.

The following is the distribution of steering angles in the Udacity dataset:



This clearly depicts that most of the dataset tends to fall in the straight line driving category that is 0 steering angle which can bias the network during training. The following distribution shows the scenario after doing augmentation in the form of shift_img function which has been mentioned earlier:



This shows that now after this step, the number of 0 steering angles have reduced to 10% of its initial count. This helped a lot in maneuvering the sharp turns.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 9. I started by using high number of epochs that is 25 and the car started with a zig zag motion as soon as it started moving which showed that it had been trained on being too idealistic and was overtrying to remain in the middle of the lane. I then reduced the epochs and it became better until I reached 9 epochs below which the car went off track which showed it was undertrained. I used an adam optimizer so that manually training the learning rate wasn't necessary.