Behavioral Cloning

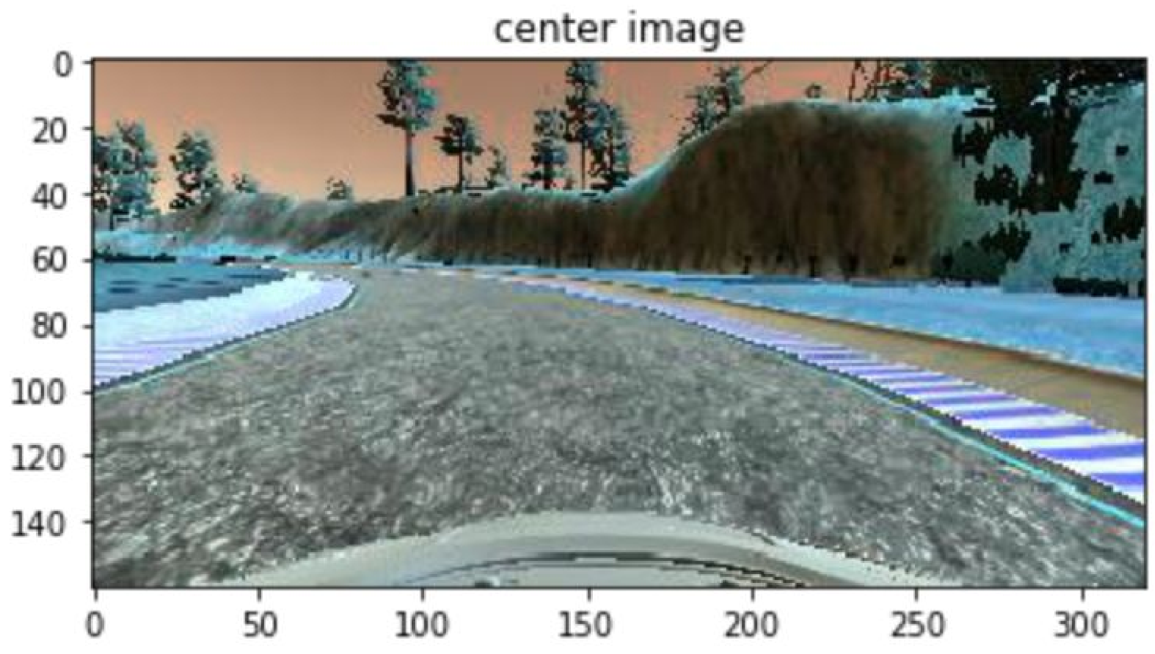
Introduction

In this project, I will use a convolutional neural network model to predict steering angles from road images. The goals/steps of this project are the following:

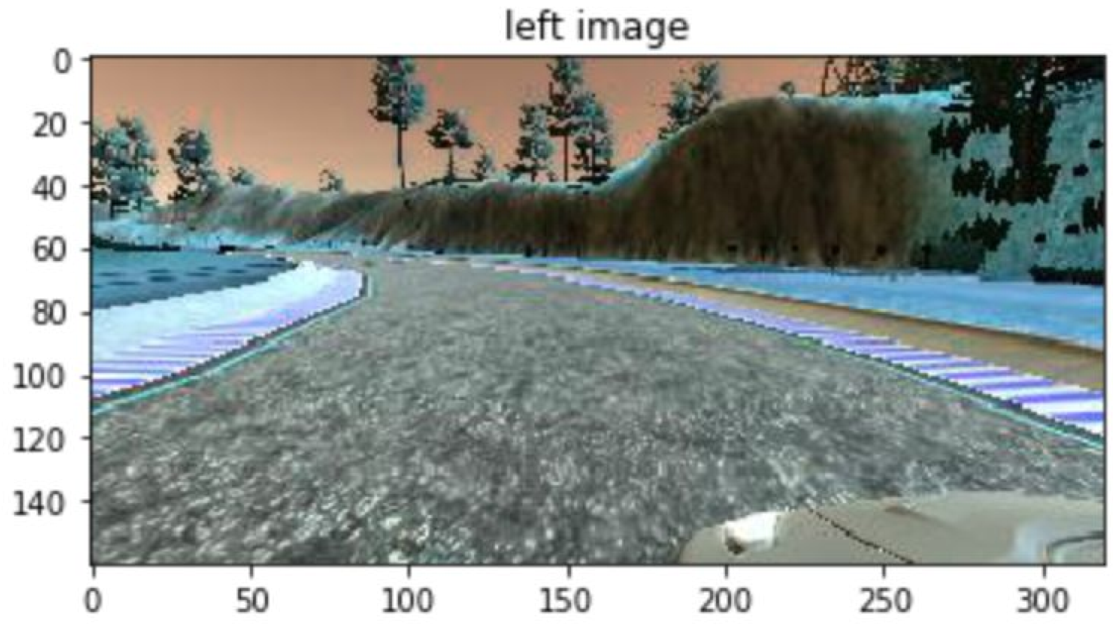
- Use the simulator to collect data of good driving behavior.
- Build a convolution neural network model in Keras that predicts steering angles from images.
- Train and validate the model with the training and validation set.
- Test that the model successfully drives around track one without leaving the road.
- Summarize the results with a written report.

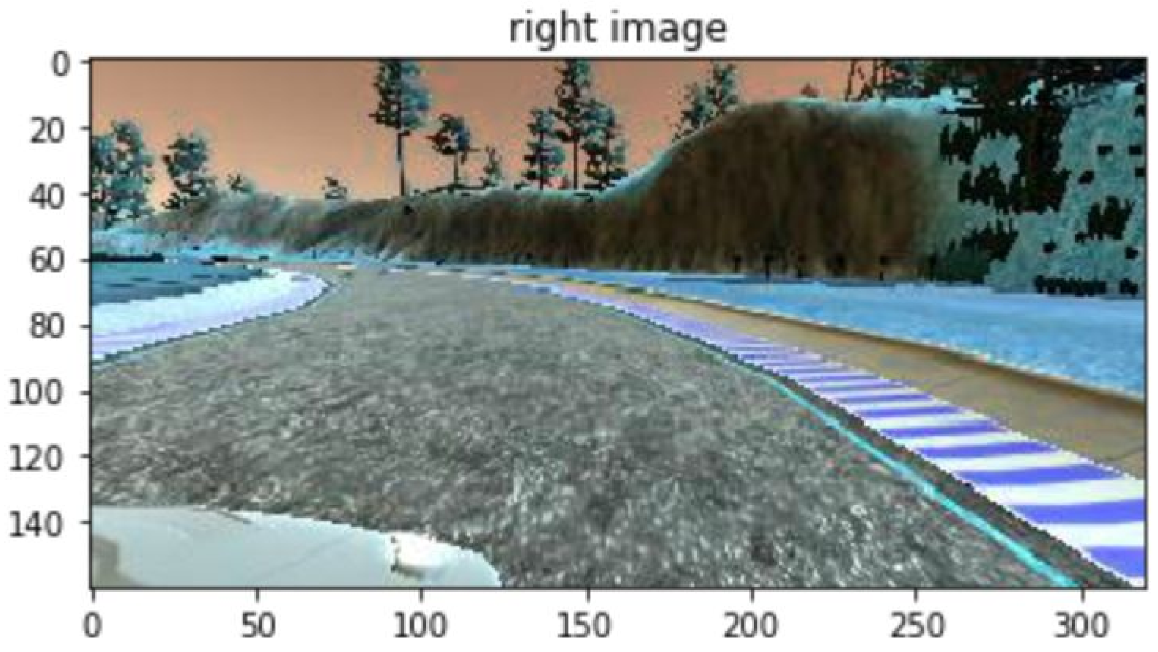
Data collection strategies

The code for this section is in ‘build_dataset.py’, and ‘training.py’(‘get_data ’, and ‘generator’ functions).
I drove a car in a simulated environment to collect images of good driving behavior. I first recorded some laps on 'track 1' the using center lane driving, here is an example.

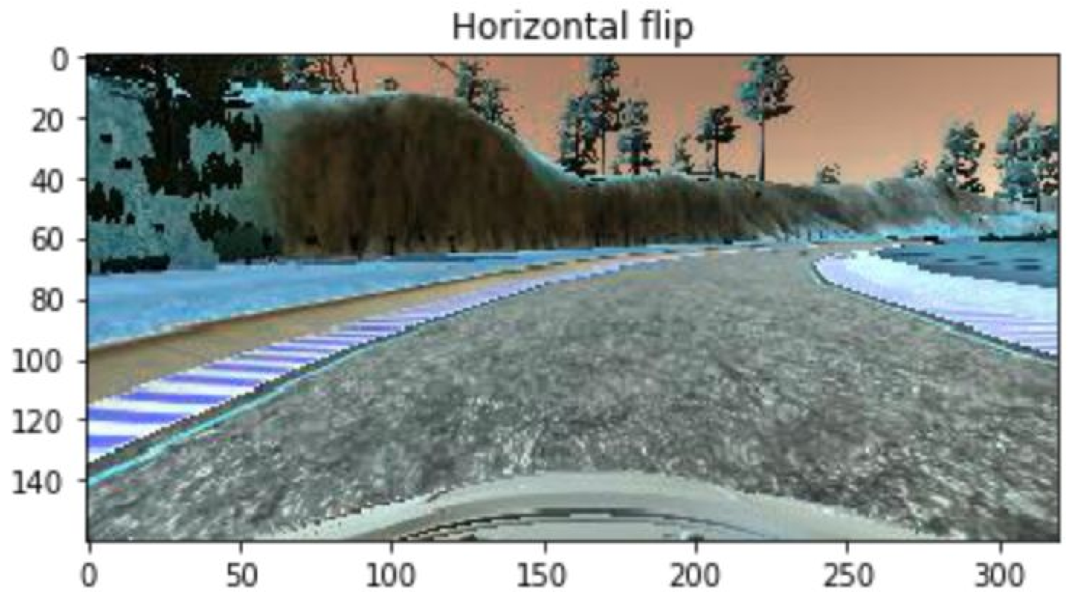


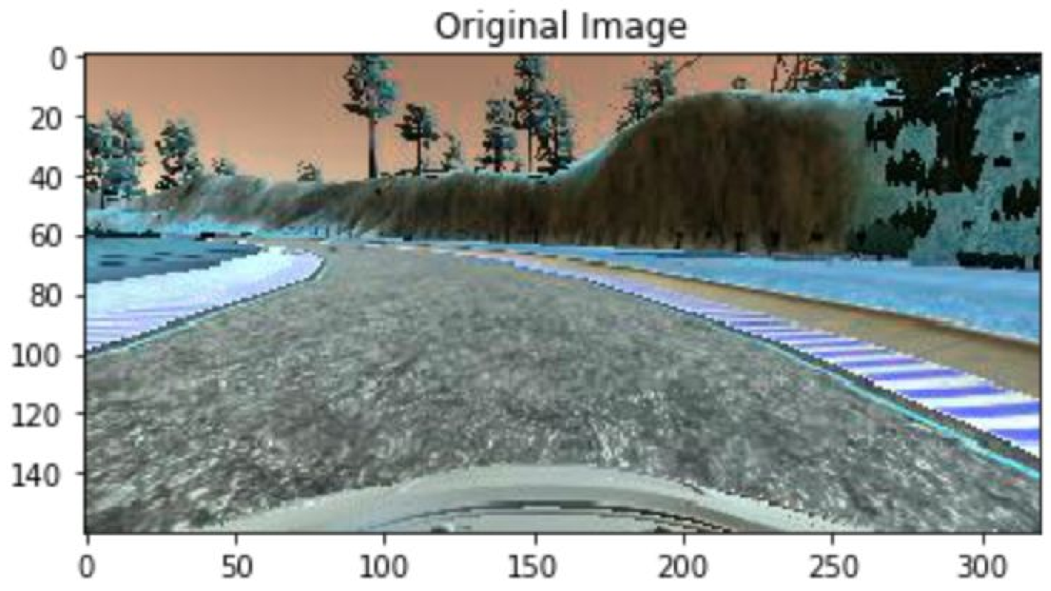
I then recorded the vehicle recovering from the left and right side of the road back to center so that the vehicle would learn to recover back to the center if the car veers off to the sides. Below are two images example.





The center images’ steering angles were used to calculate both left and right images’ steering angles. The left and right angles were calculated by subtracting and adding a value of 0.2 to the center angles respectively.
To augment the data I also flipped the images and angles thinking that this will allow my model to generalize better. For example here is an image that I flipped:





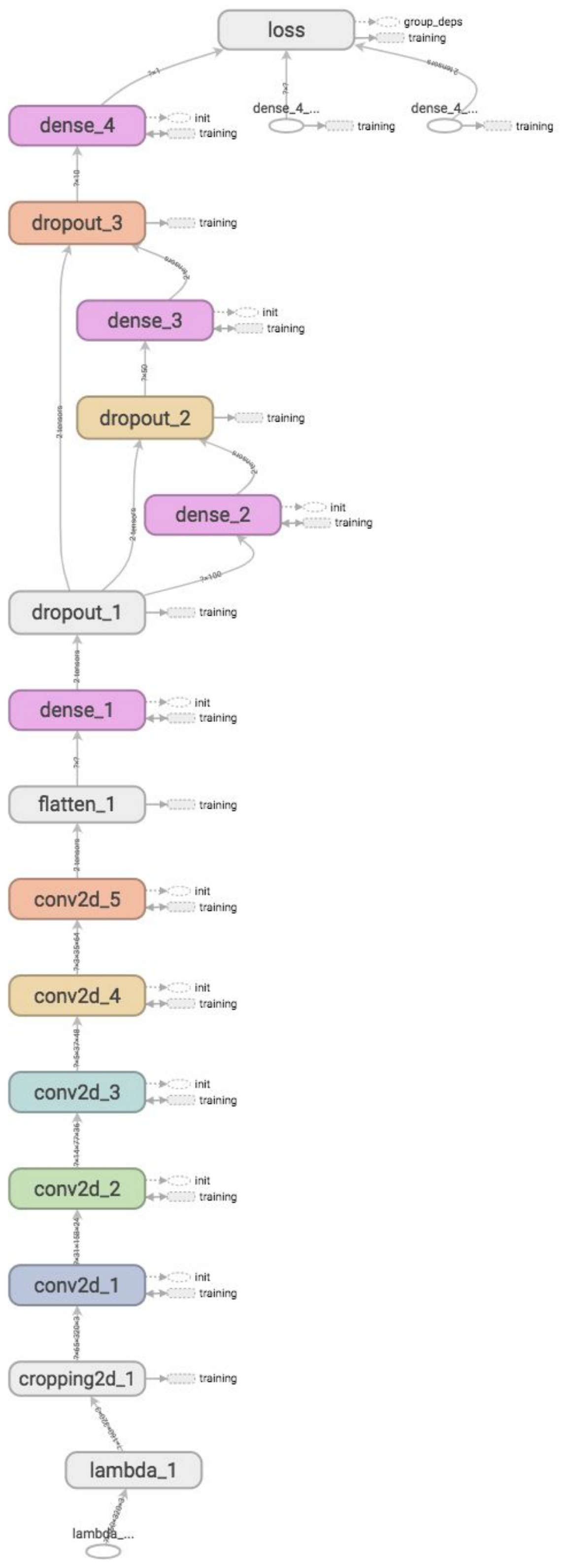
The size of the final data set is 45396, which I divided into two subsets: 80% of data for training and 20% data for validation. The validation set is used to gauge how well our model behaves on unknown data.

Model Architecture and Training Strategy

An appropriate model architecture has been employed. My model consists of the following layers:

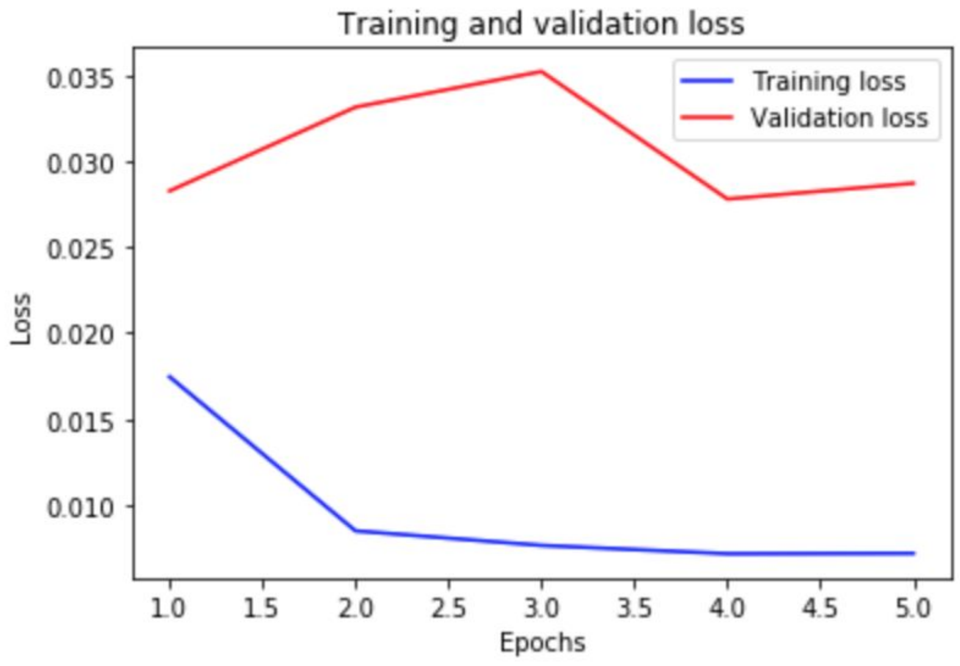
- Lambda layer: This layer is used to normalize the pixels value in images with a value between -0.5 and 0.5 and mean value of 0.
- Cropping layer: This layer crops the input image so that the car only see the road.
- Three (5x5) convolutional layers: These layers learn representations from the input image using the convolution operation followed by a ‘relu’ non-linearity function. They have an output filter value of 24, 36, 48 respectively.
- Two (3x3) convolutional layers with output value of 64, and ‘relu’ non-lineary function.
- Four fully connected layers.
- The dropout layers between the fully connected layers: This layer drops some neurons (20%) to deal with regularization.

The final architecture is in the figure below:


loss
group_deps
training
dense_4
init
training
dense_4_...
training
dense_4_...
training
dropout_3
training
dense_3
init
training
dropout_2
training
dense_2
init
training
dropout_1
training
dense_1
init
training
flatten_1
training
conv2d_5
init
training
conv2d_4
init
training
conv2d_3
init
training
conv2d_2
init
training
conv2d_1
init
training
cropping2d_1
training
lambda_1
lambda_...

Train and validation model

I trained the model with five epochs. The figure below shows the validation and training loss.



Test the model

The final model has been tested on 'track 1' and the vehicle behaves well.

Discussions

This model performs well on track 1. The vehicle stays on the road lane but not quite at the center. It also goes to the far left and the far right of the road, but each time it does it, it comes back close to the center. This behavior can be explained by the data collection, when I was collecting the data, I wasn’t always at the center of the road because I was learning to drive in the simulated environment. This can be resolved by collecting more good data road behavior.

We also have to note that this model doesn’t perform well on ‘track 2’ which has a more complex road. This can be resolved by collecting data from track 2.
Once we have good data that can generalize, we can also increase some hyperparameters such as the number of epochs.