Behavioural Cloning

The goals / steps of this project are the following:

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

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

- My project includes the following files:
- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report.md or writeup_report.pdf summarizing the results
- Final video

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

python drive.py model.h5

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network based on Nvidia model (model.py lines 145-181)

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

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 195).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane and off center lane driving, For details about how I created the training data, see the next section. (model.py line 45-106)

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to make changes to the training data and employ different networks in order to get good performance

My first step was to use a convolution neural network model similar to Lenet with additional dropout layers. I thought this model might be appropriate because it is a well-known basic model and I already have some experience with it for Project 2.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting. I added some dropout layers in order to combat overfitting. I also added additional layers for Normalizing the data and to crop the data in order to take in only meaningful data. Cropping is used to remove the sky and trees part of the image and only focus on the road.

The model produced high accuracy on validation set but failed to perform in the simulator. The car went off track at the sharp corner after the bridge.

Then I decide to use the Nvidia model as it is a known model which has already been used successfully for a similar work. I kept the additional layers for Normalizing and Cropping as same and replace Lenet architecture with Nvidia. I also multiplied the whole steering dataset by 1.5 in order to make it more responsive. This made the model oscillate a bit (specially at higher speeds) but comes in handy at sharp corners.

I also had to play around the steering modification factor in case I am using images from left or right dataset (more details in the Creation of Training Dataset and Preprocessing section) Nvidia model gave less validation accuracy than Lenet but was able to perform well on track one.

After 5 epochs, the validation loss was 0.07

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 (model.py lines 18-24) consisted of a convolution neural network with the following layers and layer sizes ...

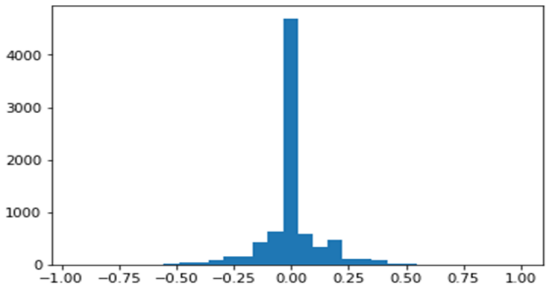
Here is a visualization of the architecture:

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 160, 320, 3)	0	lambda_input_1[0][0]
cropping2d_1 (Cropping2D)	(None, 90, 320, 3)	0	lambda_1[0][0]
convolution2d_1 (Convolution2D)	(None, 86, 316, 24)	1824	cropping2d_1[0][0]
activation_1 (Activation)	(None, 86, 316, 24)	0	convolution2d_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 43, 158, 24)	0	activation_1[0][0]
convolution2d_2 (Convolution2D)	(None, 39, 154, 36)	21636	maxpooling2d_1[0][0]
activation_2 (Activation)	(None, 39, 154, 36)	0	convolution2d_2[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 19, 77, 36)	0	activation_2[0][0]
convolution2d_3 (Convolution2D)	(None, 15, 73, 48)	43248	maxpooling2d_2[0][0]
activation_3 (Activation)	(None, 15, 73, 48)	0	convolution2d_3[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 7, 36, 48)	0	activation_3[0][0]
dropout_1 (Dropout)	(None, 7, 36, 48)	0	maxpooling2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 5, 34, 64)	27712	dropout_1[0][0]
activation_4 (Activation)	(None, 5, 34, 64)	0	convolution2d_4[0][0]
maxpooling2d_4 (MaxPooling2D)	(None, 2, 17, 64)	0	activation_4[0][0]
flatten_1 (Flatten)	(None, 2176)	0	maxpooling2d_4[0][0]
dense_1 (Dense)	(None, 1164)	2534028	flatten_1[0][0]
dropout_2 (Dropout)	(None, 1164)	0	dense_1[0][0]
dense_2 (Dense)	(None, 100)	116500	dropout_2[0][0]
dropout_3 (Dropout)	(None, 100)	0	dense_2[0][0]
dense_3 (Dense)	(None, 50)	5050	dropout_3[0][0]
dropout_4 (Dropout)	(None, 50)	0	dense_3[0][0]
dense_4 (Dense)	(None, 10)	510	dropout_4[0][0]
dense_5 (Dense)	(None, 1)	11	dense_4[0][0]

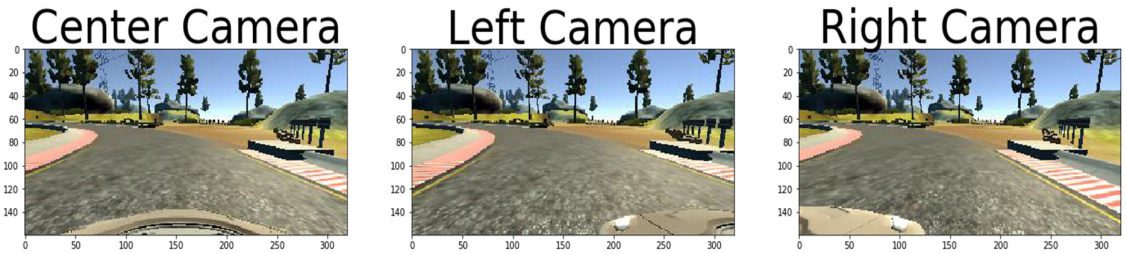
Creation of Training Dataset and Preprocessing

To capture good driving behaviour, I first started with my own data. I recorded two laps on track one while keeping the vehicle in the middle lane as much as possible. But I found it hard to do especially at few sharp corners. And the first rough pass of the model was not working at all.

So I decided to use the data provided by Udacity against my own and augment it as necessary. The car, in the first pass with Udacity data was able to do the straight line and failed at the corner. The histogram shows the udacity data. As can be seen, the data is not uniform with a lot of zero angles.



Next thing I did was to use the images from all 3 cameras – left, right and center.



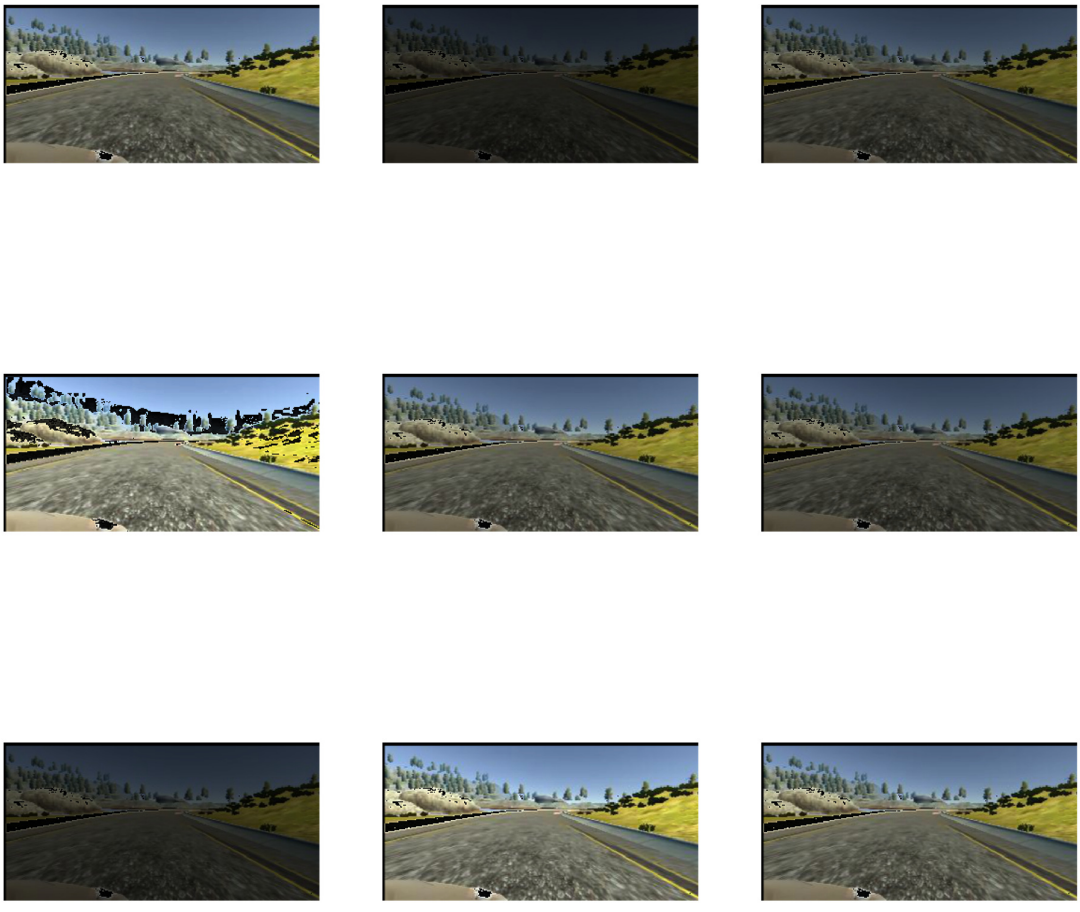
And use the left and right images as they are from center camera for training to mimic the departure from center of the lane. To compensate for it, I modified the steering angle accordingly (+- 0.25). This made the dataset about 24000 images and the training was slow. With this set, car passed the first turn and the bridge section but failed at the sharp corner just after the bridge. Also the performance took a hit as the training was slow.

I next thought of reducing the dataset and also making the dataset more uniform. So instead of taking all the left, right, center images as previously, I randomly took one of them and applied some processing to it- random brightness (to generalise the

image data) and translations in X and Y on the images and the modifications to steering angle accordingly. I used the approach defined in Vivek Yadav's blog post for this.

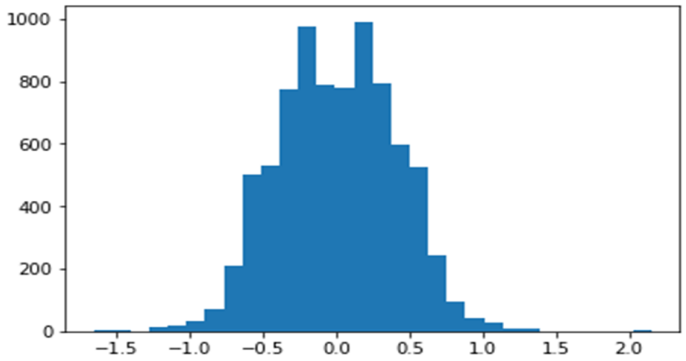
Some examples from the image processing:

Random Brightness Augmentation:

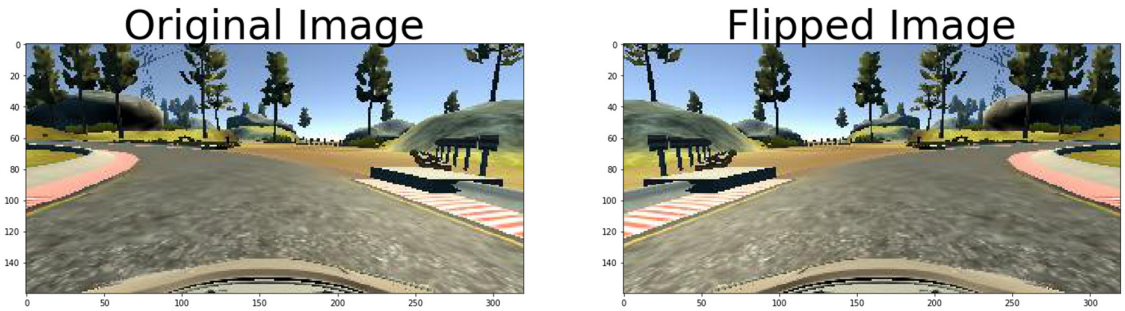


Random Translations

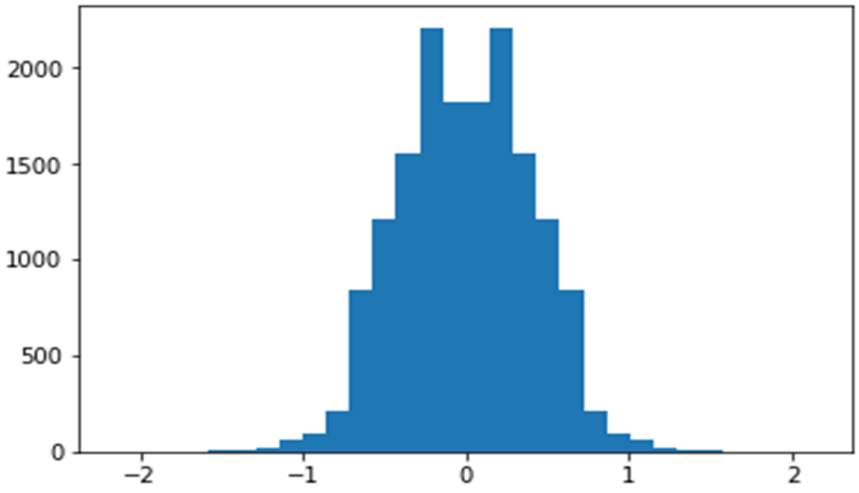
I also at this stage scale all the steering by 1.5 to make the model more responsive. This made the dataset histogram like this. There were some differences in left right of histogram.



Next I did was the flipping the image to make dataset more uniform and multiplying the corresponding steering angle by -1. Here's the example:



I finally ended up with around 16000 images with this distribution:



This is uniform and well distributed.

I then pre-processed this data by inbuilt Normalization layer in the model and cropped it with the Cropping layer.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

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 5 as anything more is not helping the model. I tried increasing the number of epochs but accuracy wasn’t improving. I used an adam optimizer so that manually training the learning rate wasn't necessary.