Behavioral Cloning

Behavioral Cloning Project

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

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

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

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

For my model I used the architecture published by Nvidia for Self-Driving Cars (https://developer.nvidia.com/blog/deep-learning-self-driving-cars/)

The implemented model summary can be seen below.

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_1 (Conv2D)	(None, 31, 158, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 77, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 37, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 35, 64)	27712
conv2d_5 (Conv2D)	(None, 1, 33, 64)	36928
dropout_1 (Dropout)	(None, 1, 33, 64)	0
flatten_1 (Flatten)	(None, 2112)	0
dense_1 (Dense)	(None, 100)	211300
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510

dense_4 (Dense)	(None, 1)	11

Network features:

- The model uses lambda layer to normalize and mean center the images.
- The cropping2d layer is used to remove the skies and hood of the car.
- The model consists of 9 layers - A normalization layer, followed by 5 Convolution layers and 3 fully connected layers.
- The first 3 Convolution layers have a kernel size of 5x5 and a stride of 2x2.
- The last 2 Convolution layers have a kernel size of 3x3 and stride of 1x1.
- The Relu activation function is used in the convolution layers to introduce nonlinearity.
- The dropout layer is used before the fully connected layers as a Regularization technique.
- The final output of the network is the desired steering angle for the car.

Graphical representation of the architecture can be seen below.

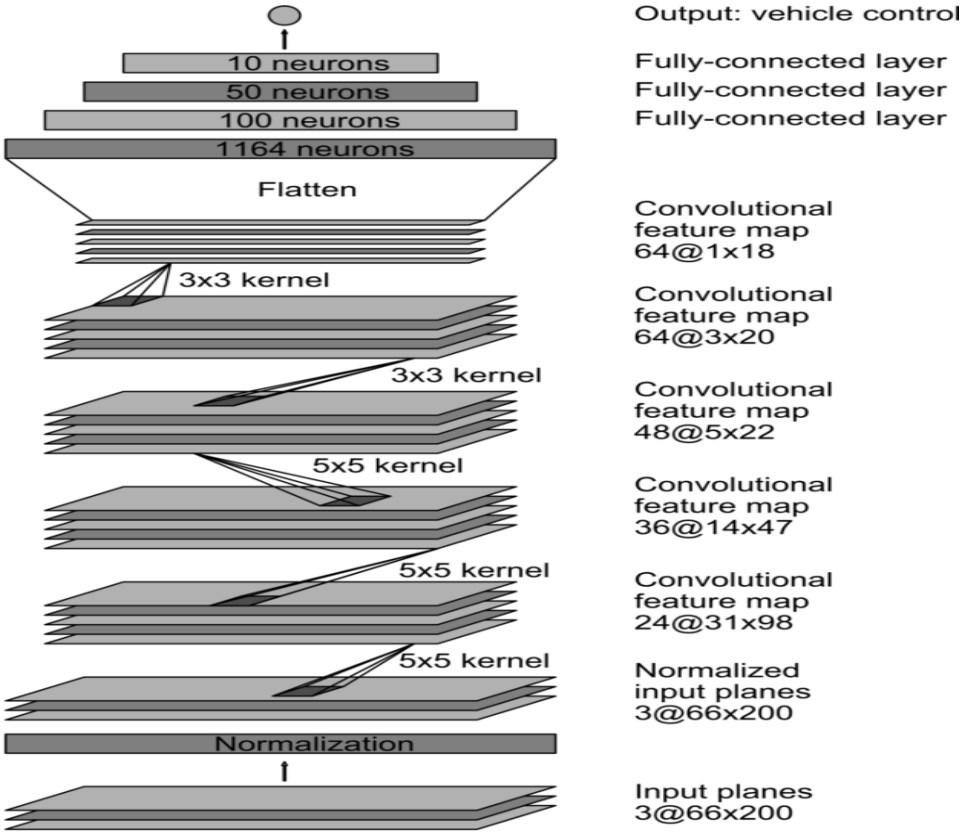

Fig 1. Nvidia CNN Architecture

2. Attempts to reduce overfitting in the model

To reduce overfitting, the model contains a dropout layer after the final convolution layer with a drop probability of 0.5 (model.py lines 45).

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 90).

The mean square loss function was used to minimize loss for steering angle.

4. Appropriate training data

I used the sample training data provided for the project and as my model performed well on the given sample data, I decided to use only the sample data the project.

Model Architecture and Training Strategy

1. Solution Design Approach and Selected Model

- First, I used LeNet architecture to train my model.
- Then, I evaluated the performance of my model on the simulator.
- After that, I flipped the images horizontally and the steering angles to augment my training data. Most of the data was taken when the car was driving counter-clockwise around the track and the vehicle was mostly turning left, so by flipping the images we can also try to remove the bias towards left turns. (model.py line 21-29)
- After, Augmenting the data, I also used the images from the left and right cameras mounted on the car. This further increased my training set data. I used a corrected steering measurement for the left and right images. (model.py line 60-78)
- Then, I cropped the images to get rid of the sky and hood of the car which can distract the model. (model.py line 88)
- After that, I implemented the Nvidia CNN architecture instead of LeNet, consisting of 5 convolution layers and 3 fully connected layers as described earlier.

- Finally, I tested my model on the simulator again and the vehicle was successfully driving autonomously around the track without leaving the road.

Note: I did not use a generator for my project as the training dataset set wasn't large enough and my model was performing very slowly with each epoch taking approximately 3 hours for a batch size of 32. With large batch sizes, the execution speeds improved, but the model failed to train well. So, Given the GPU time limitations, I did not use generators.

2. Creation of the Training Set & Training Process

As mentioned earlier, I used the sample training data provided for the project. The training data consists of images taken from 3 cameras mounted on the center, left and right of the car.

Here is an example image of center lane driving:

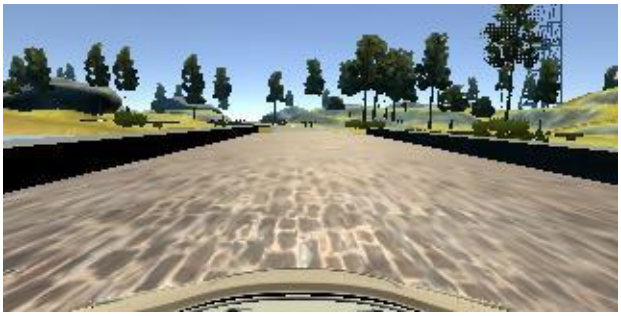

Fig 2. Center camera image

The images from the left and right sides of the car were used to teach the network how to steer the vehicle back to the center when the vehicle drifts away to the side.

Here is an example of the image taken from the left camera:

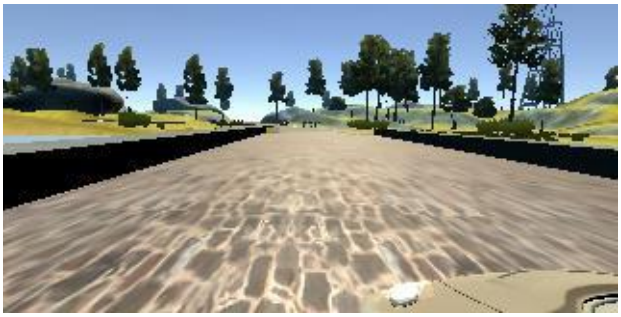

Fig 3. Left camera image

Here is an example of the image taken from the right camera:

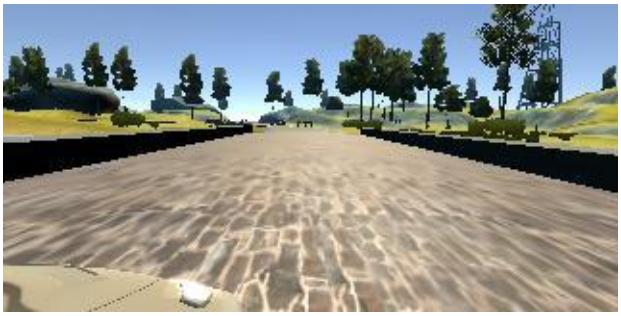

Fig 4. Right camera image

As explained earlier, I also flipped the images to augment the training data. Here is an example of the flipped image:

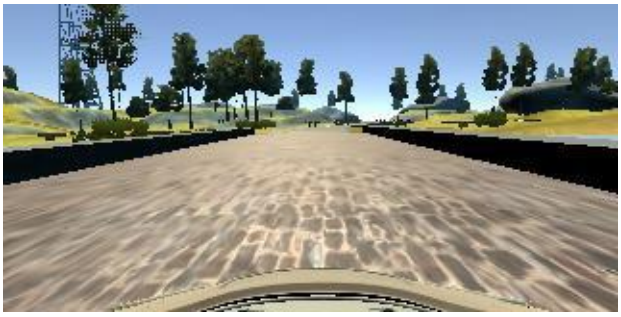

Fig 5. Flipped image

I then cropped the image, to get a portion of the image only containing the roads. Here is an example of the cropped image:

Fig 6. Cropped image

I finally trained my model using the model.fit() method into which I passed my training images and split 20% of the dataset for validation. The images were also shuffled to avoid bias while training. As the validation loss started increasing after 3 epochs, the model was trained only for 3 epochs.

A plot for the mean square loss for training and validation set over epochs is shown below:

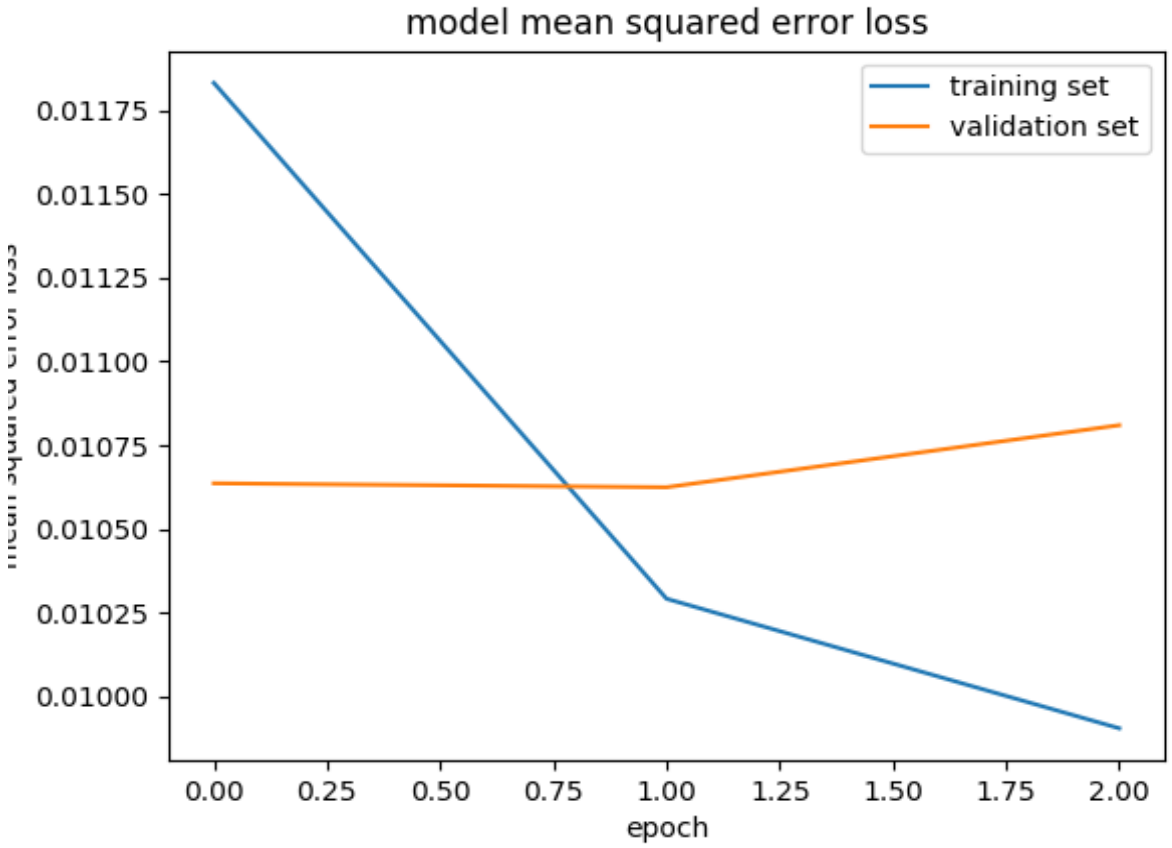

Fig 7. Mean square loss plot