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

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.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.

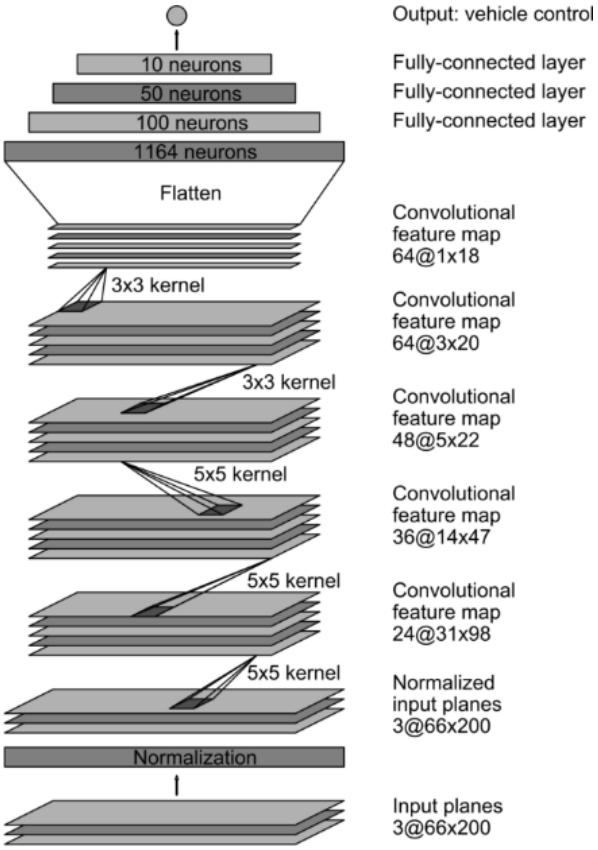
Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

I started with the basic NN similar to the lessons and got the basic setup working. Then moved to LENET before moving to NVIDIA model shown below.

Layer (type)	Output Shape	Param #
lambda_4 (Lambda)	(None, 160, 320, 3)	0
cropping2d_3 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_11 (Conv2D)	(None, 31, 158, 24)	1824
conv2d_12 (Conv2D)	(None, 14, 77, 36)	21636
conv2d_13 (Conv2D)	(None, 5, 37, 48)	43248
conv2d_14 (Conv2D)	(None, 3, 35, 64)	27712
conv2d_15 (Conv2D)	(None, 1, 33, 64)	36928
dropout_3 (Dropout)	(None, 1, 33, 64)	0
flatten_3 (Flatten)	(None, 2112)	0
dense_9 (Dense)	(None, 100)	211300
dense_10 (Dense)	(None, 50)	5050
dense_11 (Dense)	(None, 10)	510
dense_12 (Dense)	(None, 1)	11

The first layer of the network performs image normalization followed by cropping layer which takes out the sky and car section in every image. The convolutional layers are designed to perform feature extraction. I use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel, and a (1,1) strided convolution with a 3×3 kernel size in the final two convolutional layers followed by three fully connected layers, leading to a final output control value.



2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting which is added after the first 5 convolutions layers. Data Augmentation was also performed by flipping the images.

3. Model parameter tuning

The model used an Adam optimizer, so the learning rate was not tuned manually. 3 EPOCHS was sufficient to delivers good results (2% loss).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used all the 3 cameras images to ensure car learns from all 3 angles and hence drives safely. Also, I used the sample data provided in the GitHub repo for this project. I checked the images in a slide show to ensure we have a good set and was not guessing.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to start off simple, get the setup right and then move to complex models. As you can see from the IPython notebooks I went step wise understanding the nuances of each model before settling on NVIDIA model.

Preprocessing and Data Augmentation

NVIDIA model suggested normalization. I started with images/255.0 but could not converge the model to good accuracy. After a while settled on 127.5-1 which helped yield a loss of 5%. Then applied cropping layer since we need to remove the noise in each image (sky + car section). Data augmentation has been a good strategy from the last project and hence adopted by flipping the images and inverting the measurements by a factor of -1.

I faced 2 problems:-

1. Memory issue
    a. When I tried to normalize I was getting a memory issue. I didn't pay attention to the one section of the classroom initially - generators and so thought it was m AWS VM issue and scaled up to a LARGE AWS machine which solved the problem. Not recommended but worked and cost$$
2. OpenCV conversion – Initially I tried converting to YUV format (just like NVIDIA suggested) – modified drive.py but did not yield good results for some reason. Immediately roll backed the changes and focused on gathering good training data. This is the classic BGR –RGB issue which I did not pay attention until recently when I discovered that I have done everything possible and still could not get the car take a good turn. Fixing this in pre-processing step solved the problem entirely.

Create the model

Developed a replica of the NVIDIA model. Added a drop layer to prevent any over fitting (standard practice).

Evaluate the model

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 model had a low mean squared error on the training set and on the validation set. I was convinced it was good enough to be taken on-road.

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