Behavioral Cloning

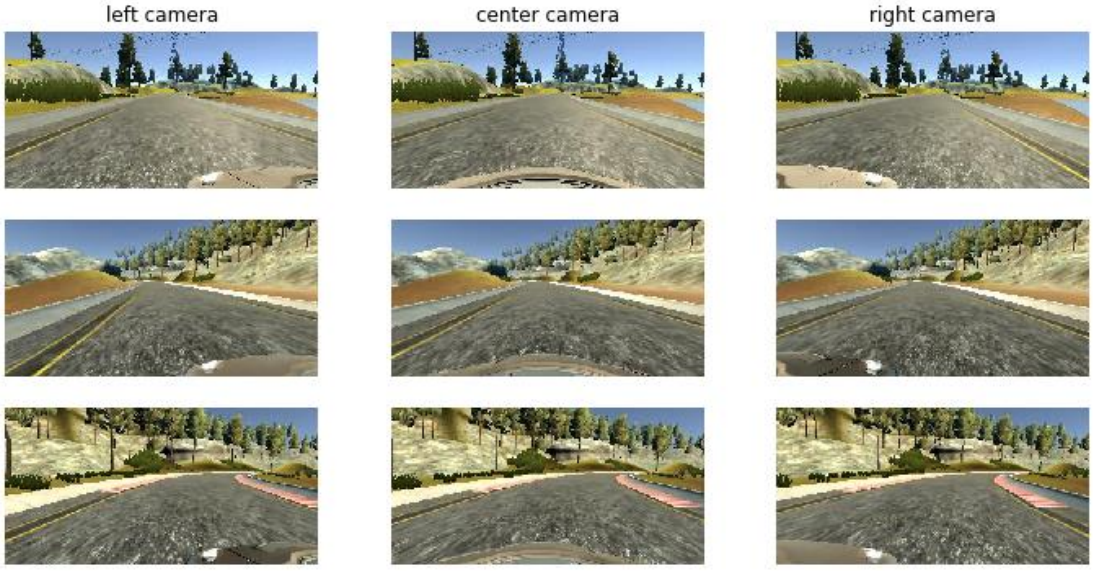
1. Behavioral Cloning Project

1.1 Files Submitted.

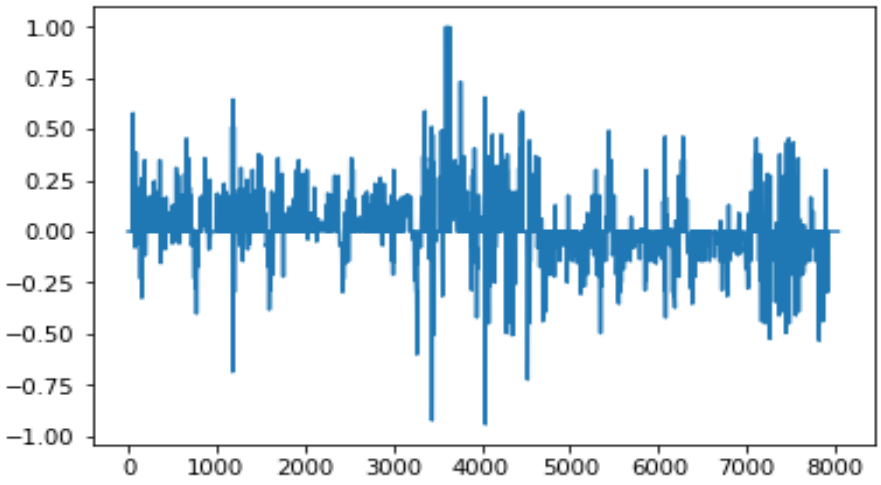
- model.py - The script used to create and train the model.
- drive.py - The script to drive the car. modify the speed to 30.
- model.h5 - The saved model.
- Writeup report as a pdf file. It explains the structure network and training approach.
- run1.mp4 - A video recording of vehicle driving autonomously 2 laps around the track 1.

1.2 Exploratory visualization of the dataset.

Examples of images from the dataset



Left, center and right camera images



Steer measurement for center camera image

1.3 Running the model

The Keras model with the weights being stored in model.h5. Run the model using the command:

$ python drive.py model.h5

2. Model Architecture and Training Strategy

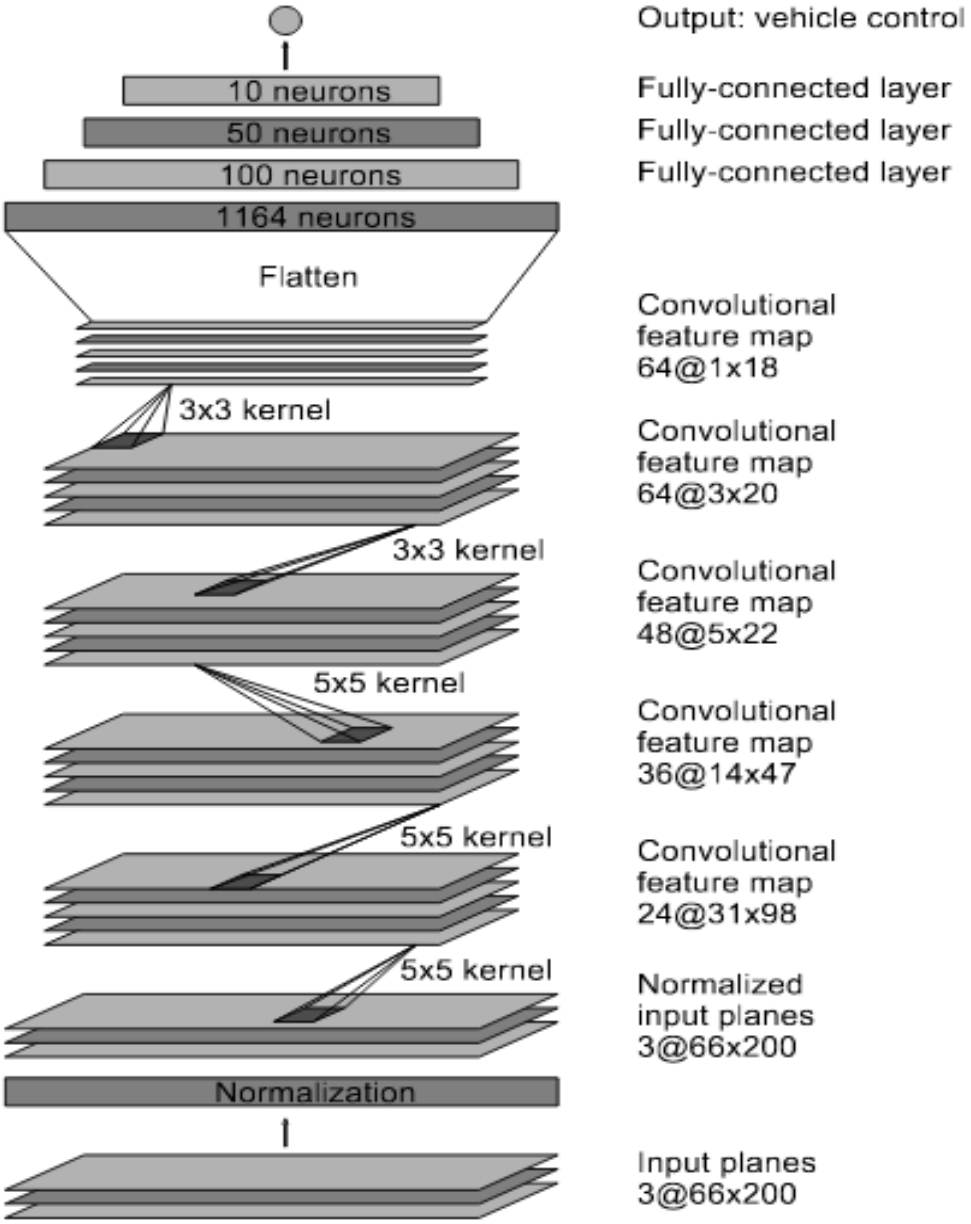
2.1 Neural Network Architecture

After reviewing the literature, the NVIDIA architecture was chosen to solve the problem. The overall NVIDIA architecture is displayed below:

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

Convolutional layers are followed by 3 fully-connected layers: finally, a last single neuron tries to regress the correct steering value from the features it receives from the previous layers.

It has been test on the road on a much more open ended terrain (public roads) and empirically demonstrated its feasibility, so it should be able to beat a simulator with ease.



NVIDIA model architecture overview

Layer (type)	Output Shape	Param #
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
lambda_1 (Lambda)	(None, 66, 200, 3)	0
lambda_2 (Lambda)	(None, 66, 200, 3)	0
conv2d_1 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 20, 64)	27712
conv2d_5 (Conv2D)	(None, 1, 18, 64)	36928
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 100)	115300
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0

NVIDIA model architecture implementation

After some experiments and comparison, I decided not to modify the model by applying regularization techniques like Dropout or Max pooling. Instead, I decided to keep the training epochs low: only three epochs. In addition to that, I split my sample data into training and validation data. Using 80% as training and 20% as validation. This can be seen at this part of the code.

Surprisingly, if apply dropout at the end of network layers, the car will drift off the side when drive to the 3rd and 4th turns, which need to be further tested and investigated.

2.2 Training Strategies

From the driving_log.csv file given by Udacity course resource, there are 8036 steer measurements and 8036 images for center, left and right cameras respectively, in order to achieve more generalize training model, below methods are used to expand the dataset:

1. Use side camera images to expand training set by 3 times, Benefits:
   1) more data to train
   2) teach network to steer back to center if drift off the side

   0.2 of correction factor is biased based on center steer, and total image and steer measurements used for the training is 8036x3=24108 respectively

2. Data augmentation to teach the car to steer clockwise and counter-clockwise by flipping images horizontally and invert steer angles. Benefits:
   1) more data to train
   2) data is more comprehensive

3. Create Pre-Processing Layer before NVIDIA model architecture to better fit the input data requirements, including:
   1) Crop 70 pixels from the image top and 25 from the bottom to remove the irrelevant information
   2) Resize mage to (66,200) for Nvidia model
   3) Normalize image to [-0.5, +0.5] to better generalize

   Pre-process steps are taken inside the model to better apply to new input images

4. Indefinitely yield batches of training data. Instead of storing the pre-processed data in memory all at once, using a generator can pull pieces of the data and process them on the fly only when you need them, which is much more memory-efficient.

   Generator like a coroutine, a process that can run separately from another main routine. Instead of using return, the generator uses yield, which still returns the desired output values but saves the current values of all the generator's variables. When the generator is called a second time it re-starts right after the yield statement, with all its variables set to the same values as before.

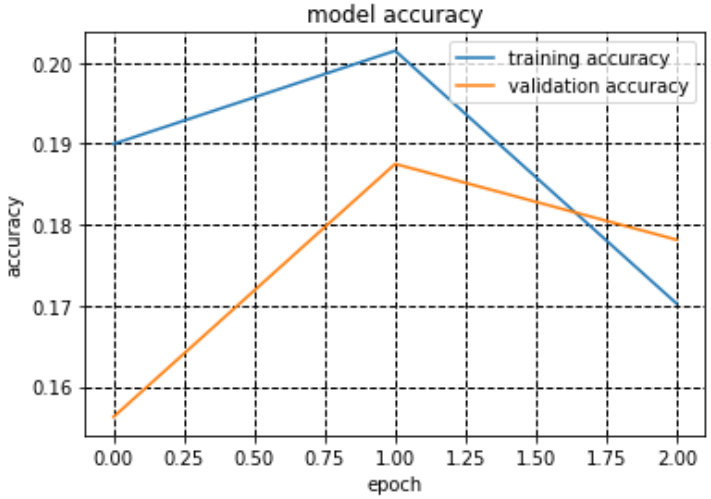
5. Splits validation from total train data by the portion of 20%, and the model uses RELU activation to reduce overfitting.

2.3 Training Result

Parameters:

Batch_size = 512, Epochs = 3, loss=mse, optimizer = Adam(lr = 0.0001)

Training and validation loss across 3 epochs



Training and validation accuracy across 3 epochs

2.4 Simulation video output

video recording simulation resolution (320x240):

$ python drive.py model.h5 run1

Creates driving video based on images stored in the run1 directory (default FPS is 60):

$ python video.py run1

3. Further discussion

1. The performance is not qualified to autonomously drive the whole lap for track 1 if introduce dropout layer in the Nvidia model, further investigation is required

2. The car is doing a bit Tarzan swings in the straightaway like drunk driving, and the concussion is increasingly accumulated until drift off the road as speed increase, further investigation is required.

3. CV2 load images in BGR colour space, the performance can be improved by adjusting the brightness of the image to make the model better generalized. Given that the track 2 is darker then track 1, this can be done by colour change BGR2HLS, increase the brightness, then HLS to RGB

4. The speed can be modified in drive.py file and the simulator will drive the car in a constant speed. However, in the real scenario, driver will lower the speed when approaching to the turn, one way to improve the performance is to include throttle as image output to train.

5. Other effective small model is welcome to be suggested since the Nvidia model is mainly for the real scenario with multiple inputs and outputs.