BEHAVIORAL CLONING

Author: Jumana Mundichipparakkal

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

INTRODUCTION

The objective of the project is to apply deep learning principles to drive a car autonomously taking an end-to-end approach of mapping camera images to steering commands for a car. We train, validate and test this in a driving simulator environment provided as part of the course.

This is a supervised regression problem, trying to map the images taken from three sets of camera (center, left and right) and steering angles taken. Input data is provided as a set of images with such steering angle labels.

SUBMISSION DETAILS

1. Files submitted

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
- behavioral_cloning.pdf summarizing the results

2. How to drive car Autonomously?

Using the Udacity provided simulator and drive.py file, the car can be driven autonomously around the track by executing. Input to the drive code is the model.h5 file containing the trained convolutional network for this project.

```
python drive.py model.h5
```

3. Neural network training code used to train the CNN for this project.

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

I started off with the NVIDIA model for training the network initially. Their CNN architecture is proven successful for this supervised regression problem. Figure 1 shows the Nvidia Model. The architecture consists of 9 layers:

1- Normalization layers- helps for faster computing with GPU
2- 5 convolutional layers- for feature extraction
3- 3 fully connected layers- targeted at controller for steering

The output of the network is the inverse-turning-radius.

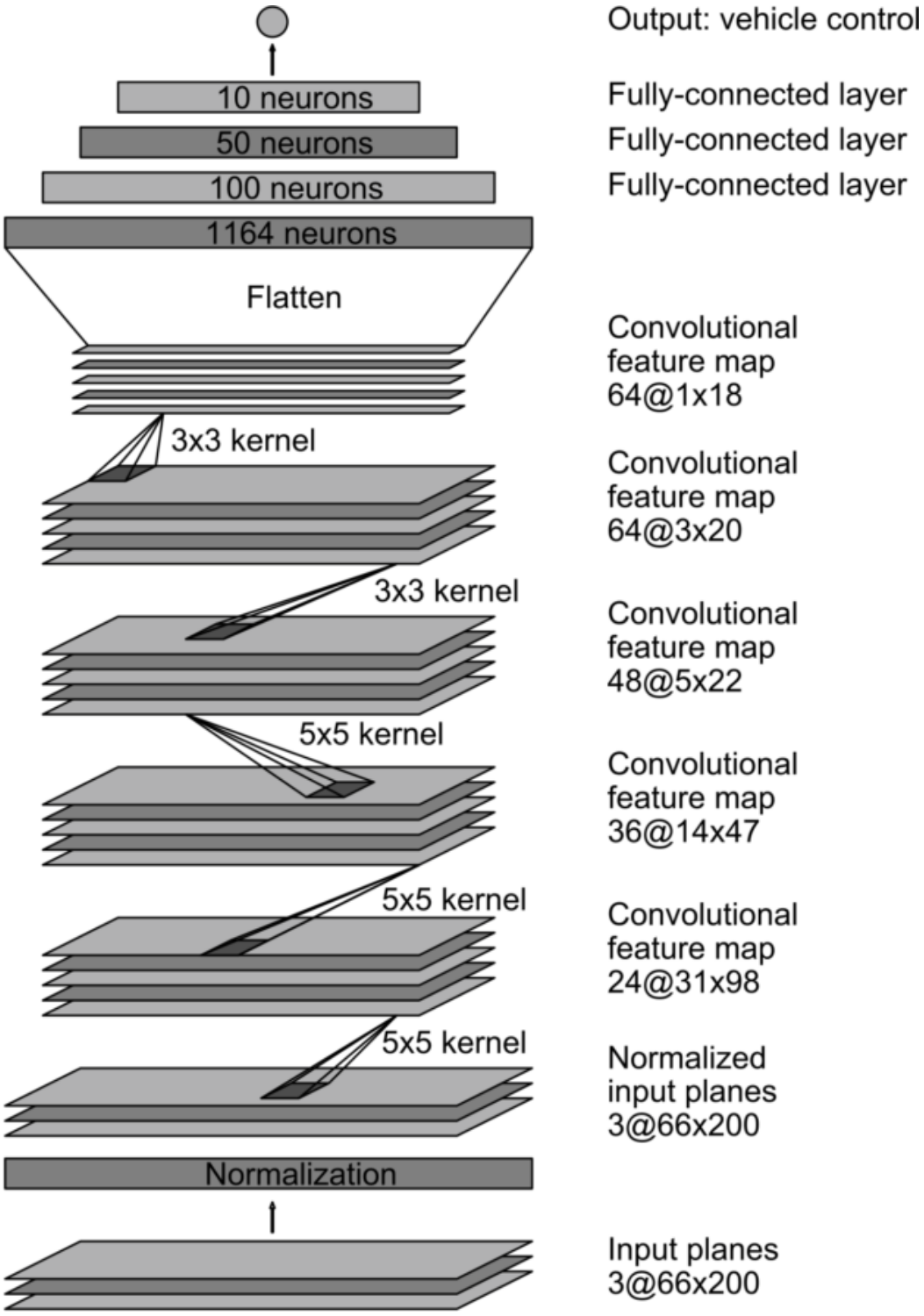

Figure 1 Nvidia CNN Architecture for end-to-end autonomous driving

1. Data Collection

Due to lack of resources and time, I decided to make use of the sample training data Udacity has provided and generate more images by processing the provided sample images. Data provided includes three images per frame taken from left, right and center cameras. Figure 2 shows an example image from different angles. There is a csv file associated with the data, that clearly indicates the path for each of these images per frame, and associated steering angle, throttle, speed and break values. As an initial step, I implemented collectImages() to collect all images in each of these angles and their corresponding steering angles to a list. For all left images steering angle is corrected by +0.25 and for right images by -0.25, respectively.

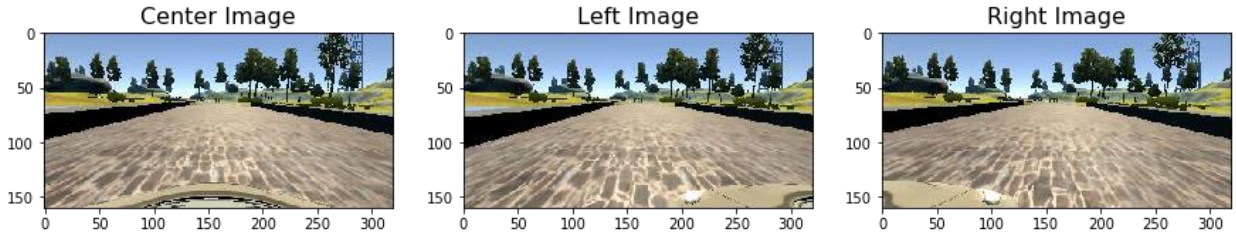

Figure 2 Example image from different cameras

2. Data Preprocessing

Images produced by the simulator in training mode are 320x160, and therefore require pre-processing prior to being fed to the CNN because it expects input images to be size 200x66. To achieve this, the bottom 20 pixels and the top 35 pixels (although this number later changed) are cropped from the image and it is then resized to 200x66, before passing to the network. Also, all the images are normalized to lie between -0.5 to 0.5.

3. Training Strategy

- The data is split into training and validation set by splitting to 80% and 20% respectively.
- For training, python generator is implemented to create a batch of 32 images. This is very important to save memory during the process. We have two generators: one for training and one for validation. I have a simple python generator that basically shuffles and randomly chooses batch size of images, in my case is 32.
- Image augmentation has been performed by adding flipped images of all the images chosen. We flip images to equalizing the number of left bent/ right bent images. The following Figure 3 shows an example of a left flipped image and Figure 4 shows an example of a right flipped image.
- Randomly choosing left right or center images during training is useful to train the recovery scenario.

4. Final Model

NVIDIA model does not use any activation functions. For preventing overfitting, I added dropouts by 0.1 in between every fully connected layer in Figure 2.

5. Training Method

I used Mean Squared Error for the Loss function to measure how close the model predicts the given steering angle for each image.

I used Adam Optimizer for optimization with it's default learning rate.

6.Testing

I tested on a lakeside track and figured out that epoch size of 5 was good enough. The model was good enough to drive around the lakeside track.

7. Outcome

I managed to get a working model for the lakeside track and video of autonomously driving car in the lakeside track is attached. I tried to run for the jungle track, but have not spent enough time to implement a solution for the same due to time limitations.

8. Dicussion

During this project, I spent a lot of time in getting the GPU work on my machine and many failed attempts of the same having not right docker file caused me a lot of pain. As I spent a lot of time in figuring out resources, I did not manage to find enough time to experiment more. Mountain track fails due to the fact that my training set data is taken from the track that I am testing on. Steps for future:

- More data collection
- Applying augmentation techniques like cropping, rescaling images etc to generate more images. I would also like to generate more data to distribute the data for different steering angles evenly for better performance.