Driving-Behaviour 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. It assumes that images are located in IMG folder in current directory.
drive.py for driving the car in autonomous mode. It is also used to record video footage from the car camera.
model.h5 containing a trained convolution neural network.
read_me.pdf summarizing the results

2. Submssion includes functional code

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

python drive.py model.h5

3. Submssion 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. There is a dedicated function for data generation, training, and testing.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

The final model that I have used consists of 3 convolutional stages (each comprised of conv, relu, and max-pooling layers) with depths 16,32, and 32 respectively; and 2 fully connected stages (one with 256 neurons and the other with 128). The model is generated in function 'generate_model'

The data is normalized in the model using a Keras lambda layer so that the values are in the range [-1 1] ((img/127.5) - 1).

2. Attempts to reduce overfitting in the model

Since we are dealing with a regression problem the validation accuracy is not an appropriate measure to see if the model is overfitting. Instead, I monitored the loss function value. If it becomes too small in the range of 1e-4 then this is a sign of overfitting. In addition, in order to assess whether the network overfits I chosen a few examples with different steering angles and checked the output of the network with respect to the true angle. This provided an indication that the network produced reasonable value. Of course, the main test was testing it on the track.

To deal with overfitting the model contains dropout layers after each convolutional layer to force the network to learn different and more general representations. Furthermore, given also the small size of the training set provided by UDACITY, I utilized the Keras built-in image generator so that each image is transformed slightly (horizontal shifts and rotations) before being presented to the network so it sees a much greater variety of data which prevents 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.

4. Appropriate training data

I used the training data provided by Udacity with augmentations and horizontal mirroring. For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to read the NVIDIA paper [1] to understand the thought process of generating such a network. It turns out that an architecture of convolutional layers followed by fully connected was used. The NVIDIA model has 5 convolutional layers and 2 fully connected (excluding the final neuron). I followed a similar architecture, however, I decided to start off with a simpler mode. of two convolutional and two fully connected since we have a smaller dataset and the test case is simpler. I tried this model with the UDACITY data as is without any augmentations. The first attempt was not successful since the car went straight on the first corner.

Then I focused on augmenting the data (random transformations) and decreased the model capacity to 3 convolutional layers and increase the neurons of the fully connected layers. At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road. I also tried it on track 2 but the car crashed after a few turns. I believe that more training data are necessary to succeed in that track.

2. Final Model Architecture

The final model architecture consisted of 3 convolutional neural network and 2 fully connected layers as shown in the summary bellow and a visualization of the architecture:

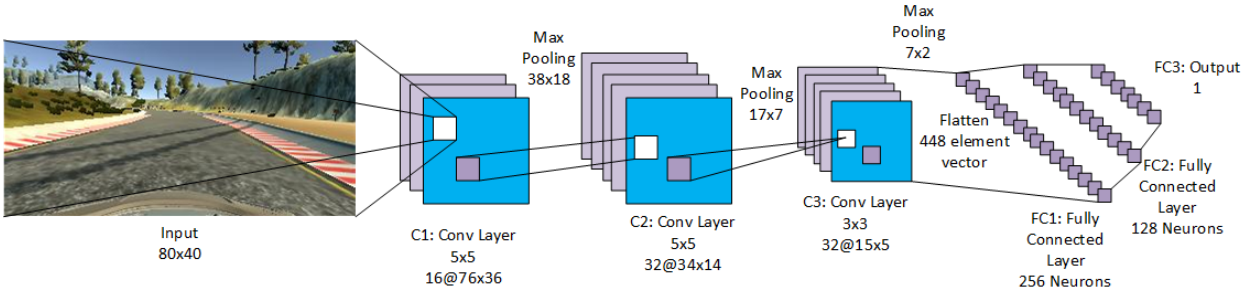

Figure 1 Architecture of the Convolutional Neural Network for behavioural cloning

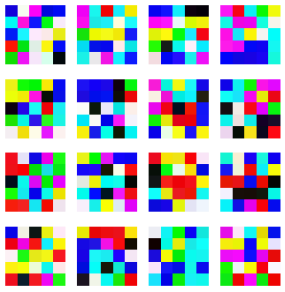

Figure 2 Visualization of the first convolutional layer filters

```
Layer (type)                 Output Shape              Param #     Connected to
====================================================================================================
lambda_1 (Lambda)            (None, 80, 40, 3)         0           lambda_input_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 76, 36, 16)    1216        lambda_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)    (None, 76, 36, 16)        0           convolution2d_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 38, 18, 16)    0           activation_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)          (None, 38, 18, 16)        0           maxpooling2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 34, 14, 32)    12832       dropout_1[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)    (None, 34, 14, 32)        0           convolution2d_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 17, 7, 32)     0           activation_2[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)          (None, 17, 7, 32)         0           maxpooling2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 15, 5, 32)     9248        dropout_2[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)    (None, 15, 5, 32)         0           convolution2d_3[0][0]
____________________________________________________________________________________________________
maxpooling2d_3 (MaxPooling2D)    (None, 7, 2, 32)      0           activation_3[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)          (None, 448)               0           maxpooling2d_3[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)          (None, 448)               0           flatten_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)              (None, 256)               114944      dropout_3[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)    (None, 256)               0           dense_1[0][0]
____________________________________________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0           activation_4[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)              (None, 128)               32896       dropout_4[0][0]
____________________________________________________________________________________________________
activation_5 (Activation)    (None, 128)               0           dense_2[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)              (None, 1)                 129         activation_5[0][0]
====================================================================================================
Total params: 171265
____________________
```

3. Creation of the Training Set & Training Process

I started out using the UDACITY dataset provided in the project resources. For all my experiments, I used only the center images provided within the csv file. I used the RGB information from the images, and scaled them between -1 and 1. Also I resize the input images to 80x40 to reduce training time and to cope with the limited memory on the GPU.

I followed different approaches with regard to generating data, however, the approach that worked for me is as follows. By observing the distribution of the data in the image shown below one can observe

that almost half the images correspond to zero steering angles. This was also pointed out by my fellow peers in the slack discussions. The approach that I followed was to balance the data a bit was to increase the amount of non-zero images. I did this by flipping those images and changing the sign of their angles. Also, I discarded 2000 images corresponding to zero-angle images. The total number of training samples is 9711. Dropping less than this amount caused the car to understeered in tight corners. In addition, I generated the data for each epoch using the Keras image generator shown below that performs random shifts and rotations on the images at each batch. I chose 64 images per batch due to compensate for GPU memory but still achieve fast training.

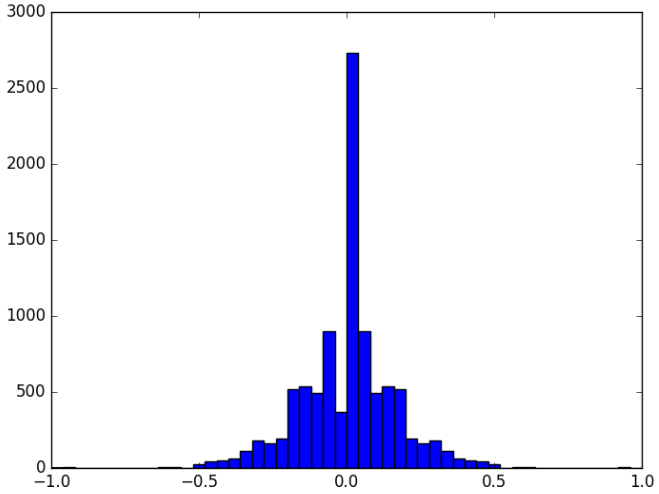

Figure 3 Angle Distribution after discarding 2000 zero angle images and flipping non-zero angle images.

I finally randomly shuffled the data set before training.

I used this training data for training the model. The ideal number of epochs for me was 10 as when trying more epochs the loss did not decrease. I used an adam optimizer so that manually training the learning rate wasn't necessary.

References

---

[1] End to End Learning for Self-Driving Cars, NVIDIA