Udacity Behavior Cloning Project

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

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

The model architecture is based on the Nvidia CNN architecture, see below:

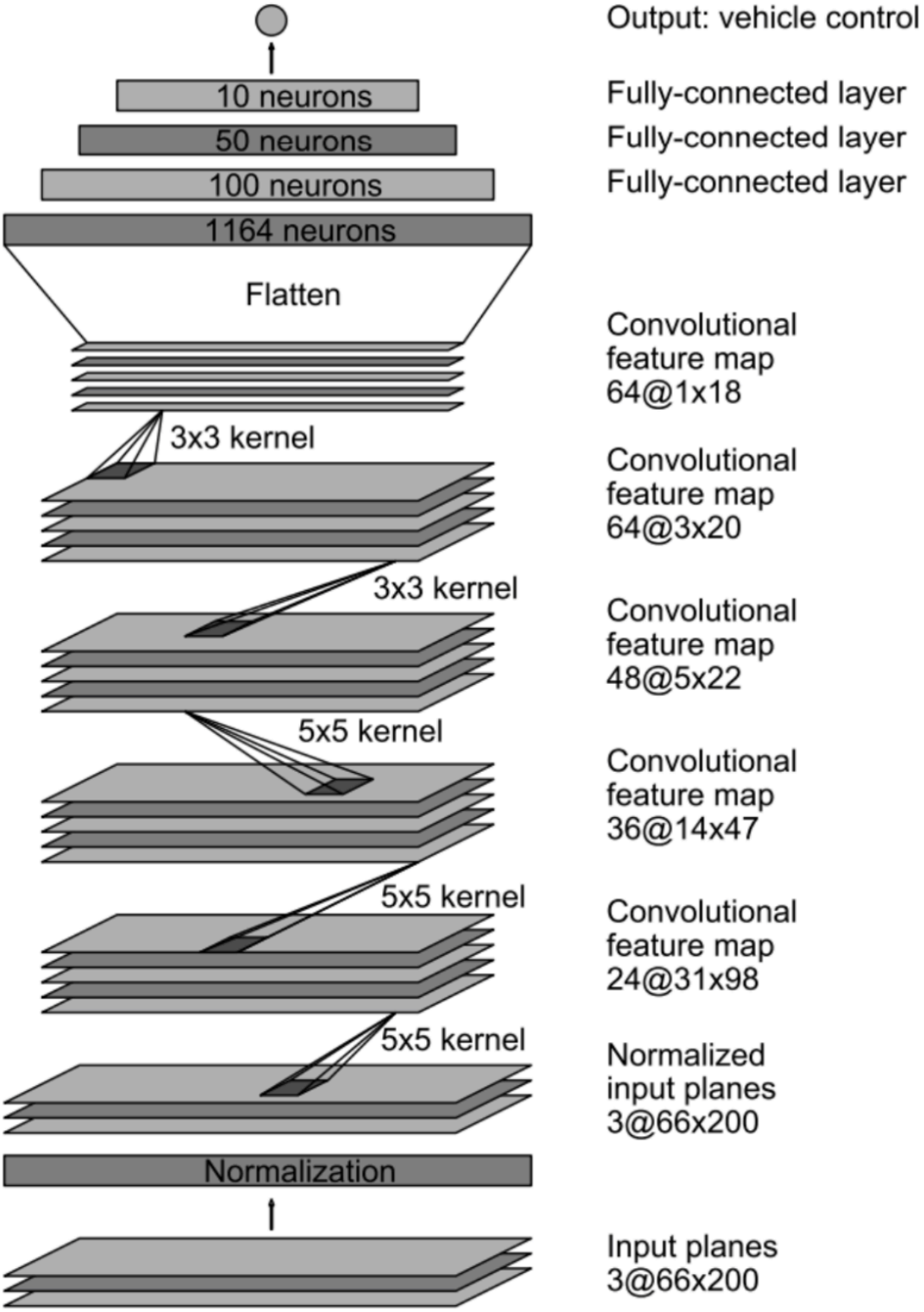


Figure 1 Nvidia CNN architecture

2. Attempts to reduce overfitting in the model

The split of test data and validation data is 1:4, and epoch is set to be 5 to reduce overfitting.

The model was trained and validated on different data sets to ensure that the model was not 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. At first, I use 0.2 for steering correction but it is not enough because the car went off the track. After trail and error, finally I used the steering correction of 0.3 to train the model and the model went on well.

4. Appropriate training data

I used the sample data provided by Udacity for this project because it's more convenient and the quantity and quality of the data is also good enough for this project. All of the center, left and right images are used to train the model.

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

I use a convolution neural network model similar to the Nvidia CNN network I thought this model might be appropriate because it provide an end-to-end solution. It consists of one Lambda layer to normalize the data, following by the Cropping layer to reduce image size and remove unnecessary image info. Correction factor of 0.3 and epoch of 5 is used to train the model

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

2. Final Model Architecture

The final model architecture is shown below:

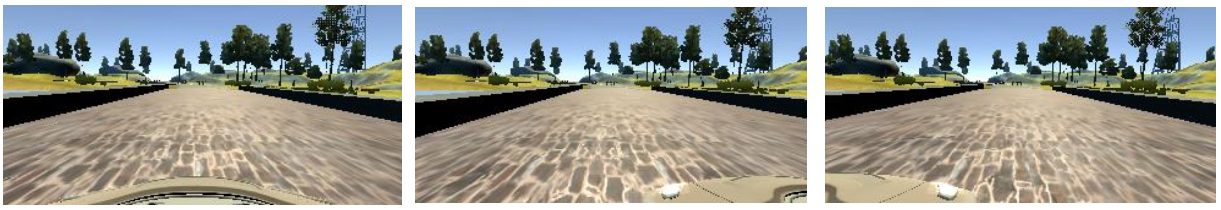
Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
convolution2d_1 (Conv2D)	(None, 31, 158, 24)	1824
convolution2d_2 (Conv2D)	(None, 14, 77, 36)	21636
convolution2d_3 (Conv2D)	(None, 5, 37, 48)	43248
convolution2d_4 (Conv2D)	(None, 3, 35, 64)	27712
convolution2d_5 (Conv2D)	(None, 1, 33, 64)	36928
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

Total params: 348,219
Trainable params: 348,219
Non-trainable params: 0

3. Creation of the Training Set & Training Process

I use the sample data provided by Udacity for this project because it's more convenient and the quantity and quality of the data is also good enough for this project. I use all the center, left and right images to train the model.

Here is an example of the center, left and right image:



center **left** **right**

To augment the data sat, I also flipped images and angles thinking that this would increase the data set. For example, here is an image that has then been flipped:

original **flipped**

I finally randomly shuffled the data set and put 20% of the data into a validation set. I used this training data to train the model. The validation set helped determine if the model was over or under fitting. I used an adam optimizer so that manually training the learning rate wasn't necessary. The figure below shows the mean square error loss of the model. We can see that the loss for training set and the validation set is decreasing as the epoch increases. Which proves that there is no obvious overfitting.