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

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
- 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

My model consists of a convolution neural network with 5x5 filter sizes and depths between 24 and 64(model.py lines 64-69)

The model includes ELU layers to introduce nonlinearity (code lines 65), and the data is normalized in the model using a Keras lambda layer (code line 63).

2. Attempts to reduce overfitting in the model

The model contains a dropout layer in order to reduce overfitting (model.py line 71).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 57). 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 (model.py line 78).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road.

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 start off with the Nvidia network shown here (https://devblogs.nvidia.com/deep-learning-self-driving-cars/)

I thought the model was appropriate because of the helpful guide in the project submission tab.

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 first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I modified the model so that I added dropout and used the elu function

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track at first, but after some fiddling I got it to work well

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 (model.py lines 62-75) consisted of a convolution neural network with the following layers and layer sizes:

```
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
________________________________________________________________
dropout_1 (Dropout)         (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
```

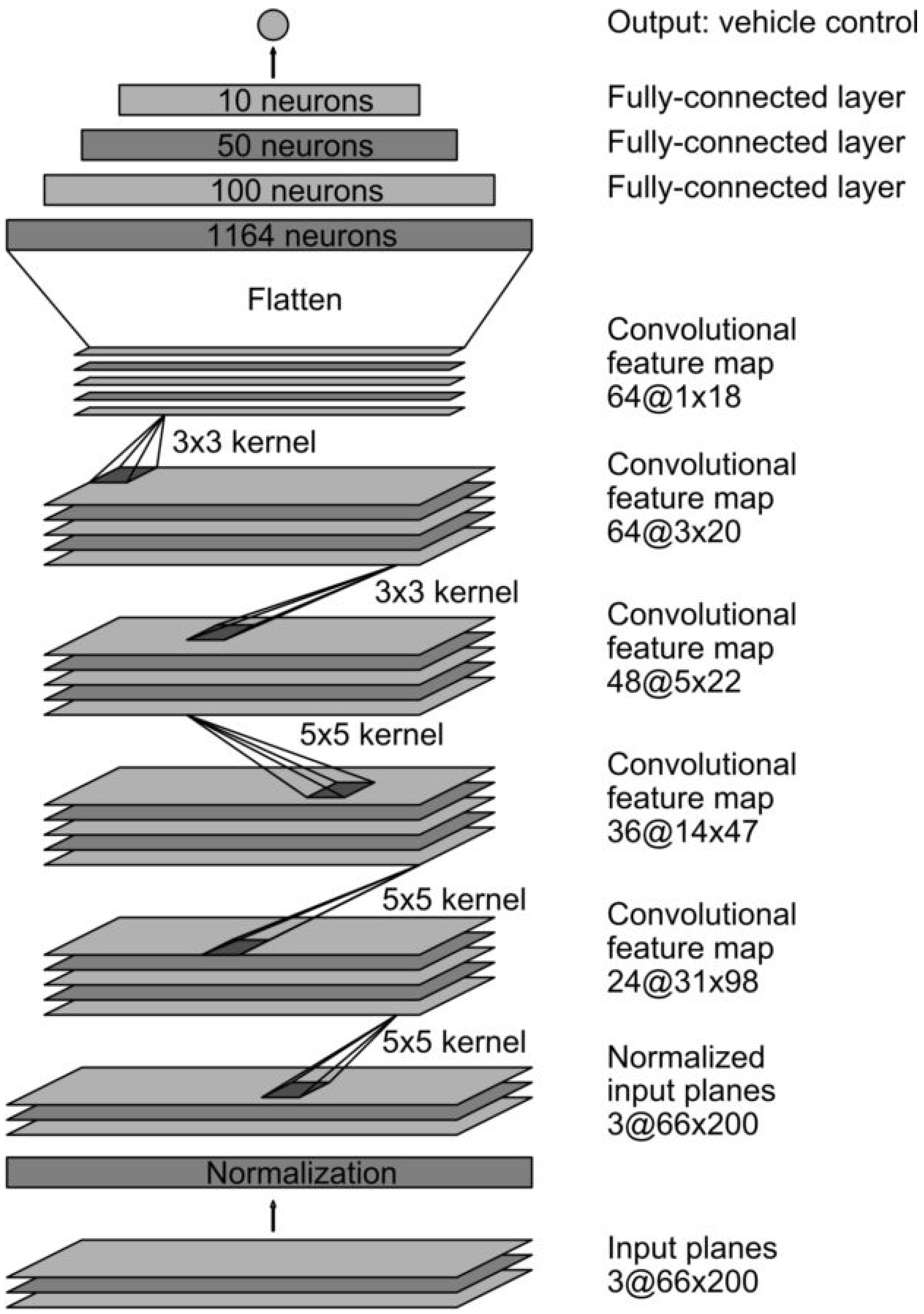
```
=================================================================
Total params: 348,219

Trainable params: 348,219

Non-trainable params: 0

_________________________________________________________________
```

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)


Output: vehicle control
10 neurons
Fully-connected layer
50 neurons
Fully-connected layer
100 neurons
Fully-connected layer
1164 neurons
Flatten
Convolutional feature map 64@1x18
3x3 kernel
Convolutional feature map 64@3x20
3x3 kernel
Convolutional feature map 48@5x22
5x5 kernel
Convolutional feature map 36@14x47
5x5 kernel
Convolutional feature map 24@31x98
5x5 kernel
Normalized input planes 3@66x200
Normalization
Input planes 3@66x200

3. Creation of the Training Set & Training Process

I managed to get the model to work with only the provided data