Behavioral Cloning

Writeup

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.md and writeup.pdf summarizing the results
- run1.mp4 video showing demo run

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

The model architecture is based on nVidia's End-to-End Deep Learning for Self-Driving Cars Model. It consists of 5 convolutional layers and 4 fully connected layers.

The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer (code line 75).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 80, 84, 88, 90).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 100). 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 110).

4. Appropriate training data

While recording training data I attempted to stay in the center on the lane while also exhibiting realistic driving behavior (e.g. curring corners). 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 test out known convolutional architecture types.

My first step was to use the Keras example mnist DNN. I thought this model might be appropriate because it was built to classify images. Yet after getting bad results with it, I switched to the original LeNet architecture (as shown in the course).

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. To combat the overfitting, I modified the model and added a dropout layer. This way I was manage to get a very usable model which almost completed the circuit. Its sole problem was identifying the dirt rode edge (see image below), which led to the car driving off-road.

I managed to solve this issue by switching to a more complex architecture, the nVidia End2End DNN as presented in the classroom and improving it with 4 dropout layers.

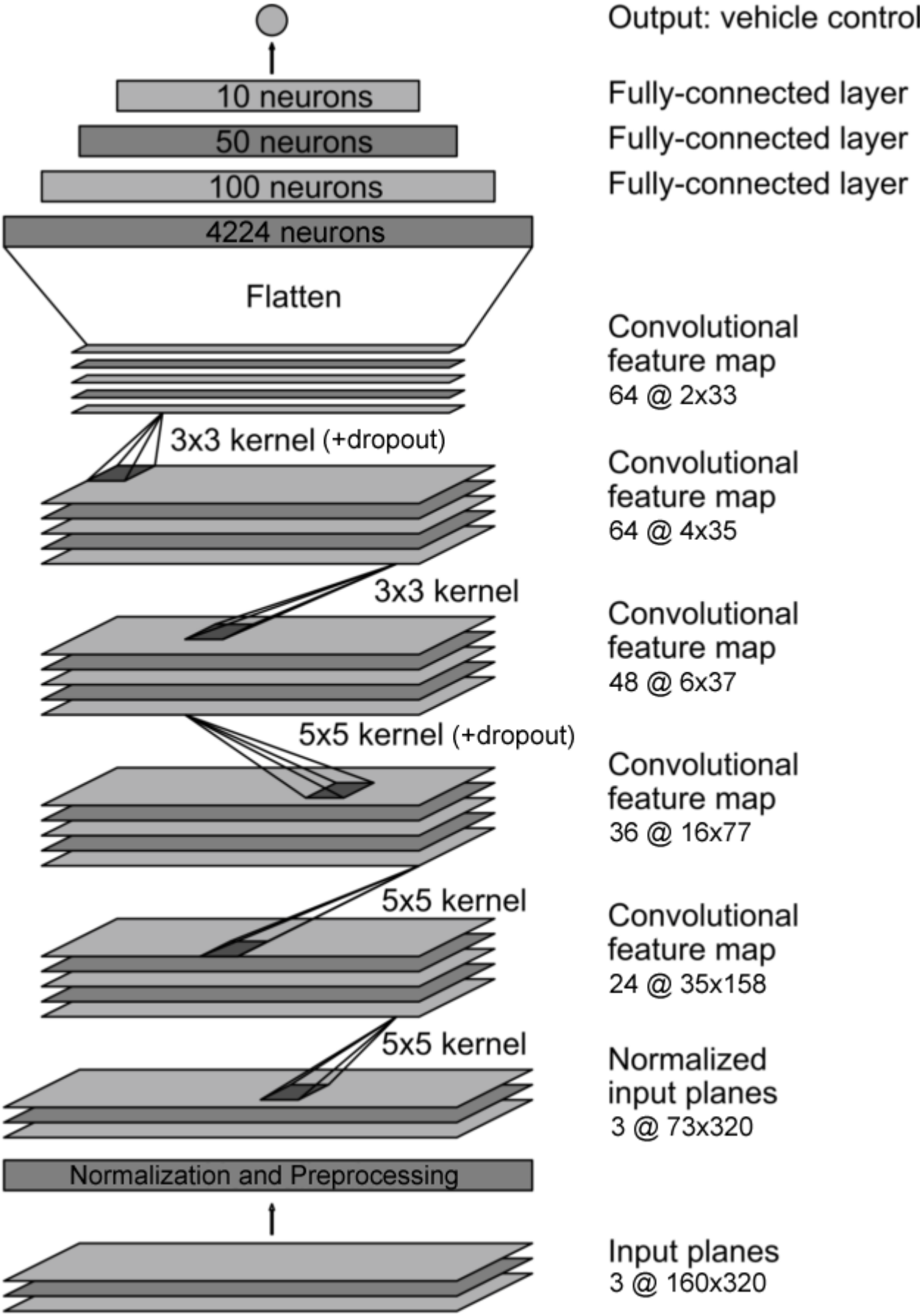
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road. The video run1.mp4 demonstrates this.

2. Final Model Architecture

The final model architecture (model.py lines 72-94) consisted of a convolution neural network with the following layers and layer sizes:

Layer (type)	Output Shape	Param
cropping2d_1 (Cropping2D)	(None, 73, 320, 3)	0
lambda_1 (Lambda)	(None, 73, 320, 3)	0
conv2d_1 (Conv2D)	(None, 35, 158, 24)	1824
conv2d_2 (Conv2D)	(None, 16, 77, 36)	21636
conv2d_3 (Conv2D)	(None, 6, 37, 48)	43248
dropout_1 (Dropout)	(None, 6, 37, 48)	0
conv2d_4 (Conv2D)	(None, 4, 35, 64)	27712
conv2d_5 (Conv2D)	(None, 2, 33, 64)	36928
dropout_2 (Dropout)	(None, 2, 33, 64)	0
flatten_1 (Flatten)	(None, 4224)	0
dense_1 (Dense)	(None, 100)	422500
dropout_3 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 50)	5050
dropout_4 (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

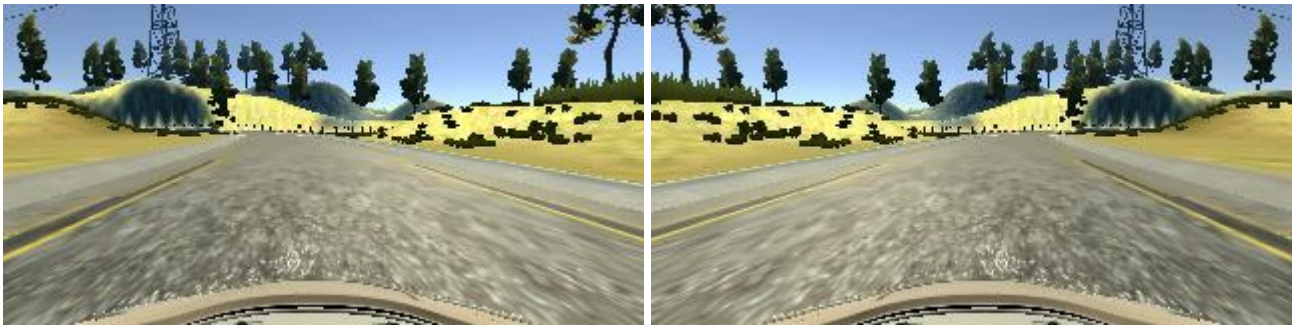
Here is a visualization of the architecture:



3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one while driving in the center of the lane, but also cutting corners from time to time. I also augmented the data set by also including the left and right

camera images as well as flipping all images (see example below). This yielded me with 10518 samples for the first track. The set was split 80-20 for training and validation.



The data preprocessing involved cropping the upper 65 and lower 22 pixels from each image and then normalizing the image between -0.5 and 0.5.

The model was trained using batches of size 64 read from the HDD with the help of a generator over 5 epochs.