Behavioral Cloning

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
- `video.mp4` contains a video of the car autonomously driving 1 lap around Track 1 (https://github.com/suvir/SelfDrivingCar-BehavioralCloning-P3/blob/master/video.mp4)
- `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

The model architecture can be found on lines 71-83 of `model.py`.

This is an implementation of the NVIDIA network architecture described here :
https://arxiv.org/pdf/1604.07316v1.pdf

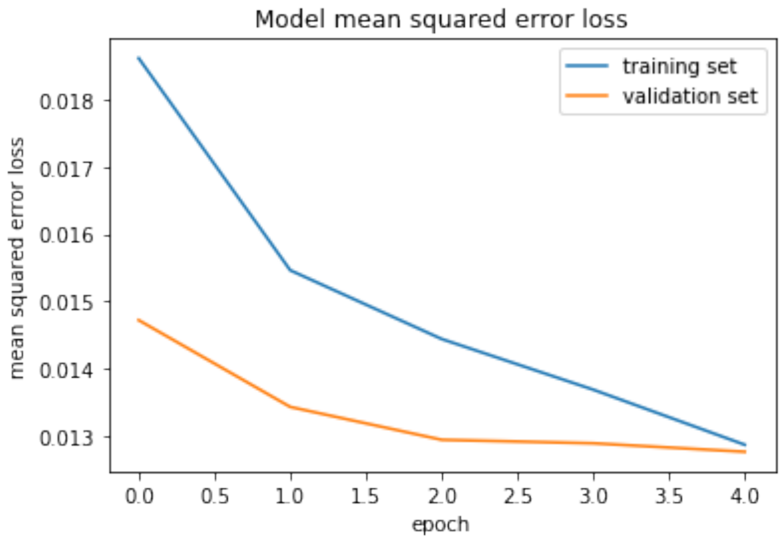
NVIDIA has employed this model for self driving cars with positive results.

2. Attempts to reduce overfitting in the model

Data was pre-processed to improve quality of input data to the learning algorith.

Also, 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.

Graph showing that training converged (and hence model did not overfit) below.



3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (`model.py` line 84).

4. Appropriate training data

Training data was a combination of images from 2 laps that I drove around Track 1 of the simulator.

Model Architecture and Training Strategy

1. Solution Design Approach

At first, I used a simple neural network architecture similar to LeNet earlier. I believe that this might work because it had worked well in the previous projects (traffic sign classification).

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. During simulation, the car was often driving off the road.

As a next step, I applied more sophisticated data preprocessing and data augmentation techniques (described later). However, the model was still not performing very well. The car was still veering off the road in many cases.

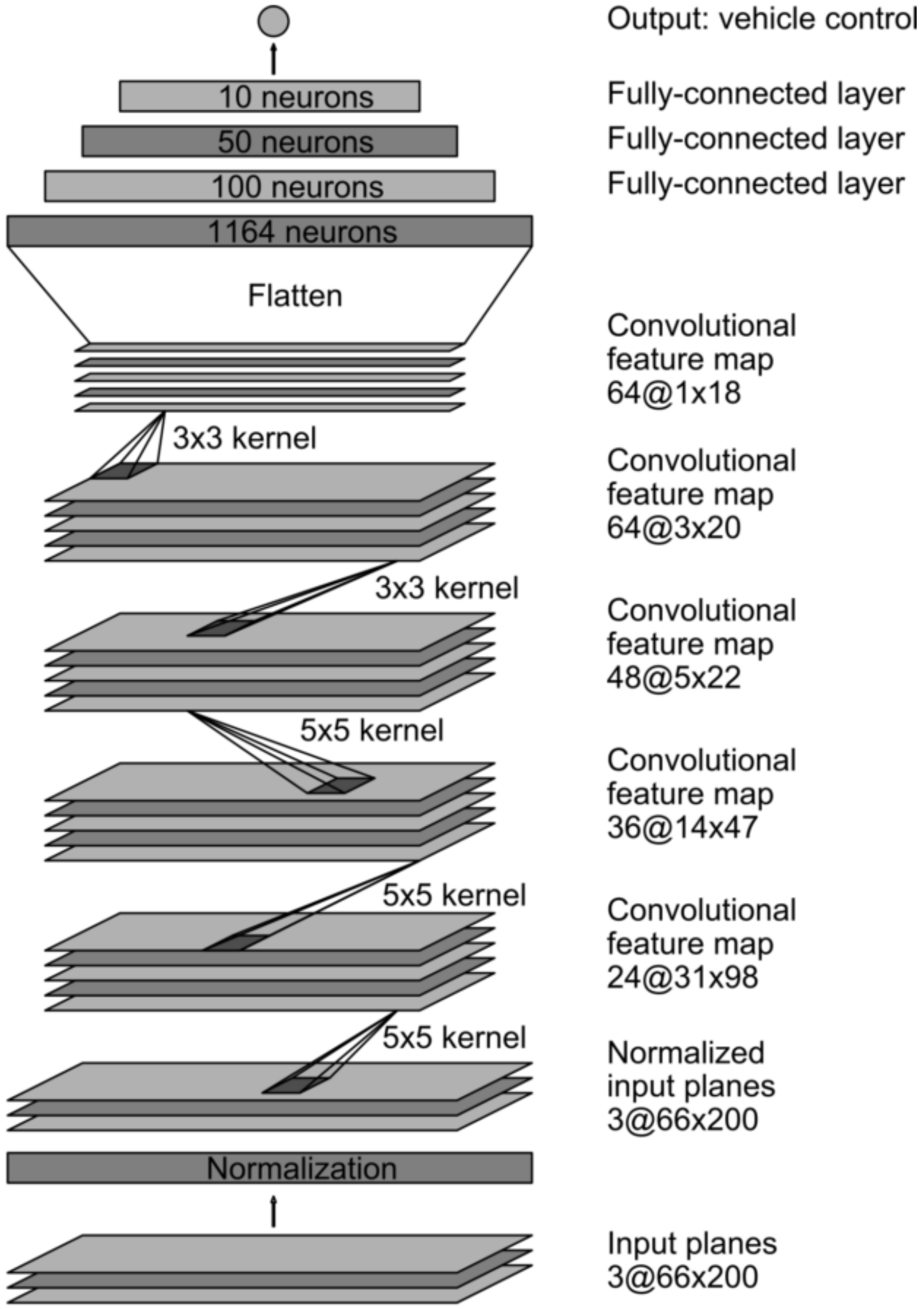
Finally, I decided to use an architecture that was proven to work well with such problems. The NVIDIA architecture described in the lectures turned out to greatly improve the results.

By changing the learning algorithm, the vehicle was able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (`model.py` lines 71-84) implements the NVIDIA architecture. It contains 5 convolutional layers and 3 fully connected layers.

Here is a visualization of the architecture from the original publication:



3. Creation of the Training Set & Training Process

To capture driving behavior, I recorded 2 laps of driving around the track. This included:

- Normal driving in the middle of the road
- Veering to the left and then steering back to the center (recovery).

Next, I preprocessed the data as follows:

- Standardized input data to have zero mean and unit variance
- Region of interest : Cropped images to remove 70 pixels from top and 25 pixels from bottom.

Next, I augmented the training data as follows:

- Flipped all the center camera images. The target measurement for this image was `-1*measurement`.
- Used left camera images. The target measurement was for these images was `measurement+0.2`
- Used right camera images. The target measurement was for these images was `measurement-0.2`
- Together, this quadrupled the amount of training data.

After the collection process, I had approx. 6,000 data points. Augmentation increased this to approx 24,0000 data points.

Finally, I randomly shuffled the data set and put 20% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 5. I used an adam optimizer so that manually training the learning rate wasn't necessary.