Behavioral Cloning

Submited by : Rajaneesh Mutyalapati

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.md or 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
```

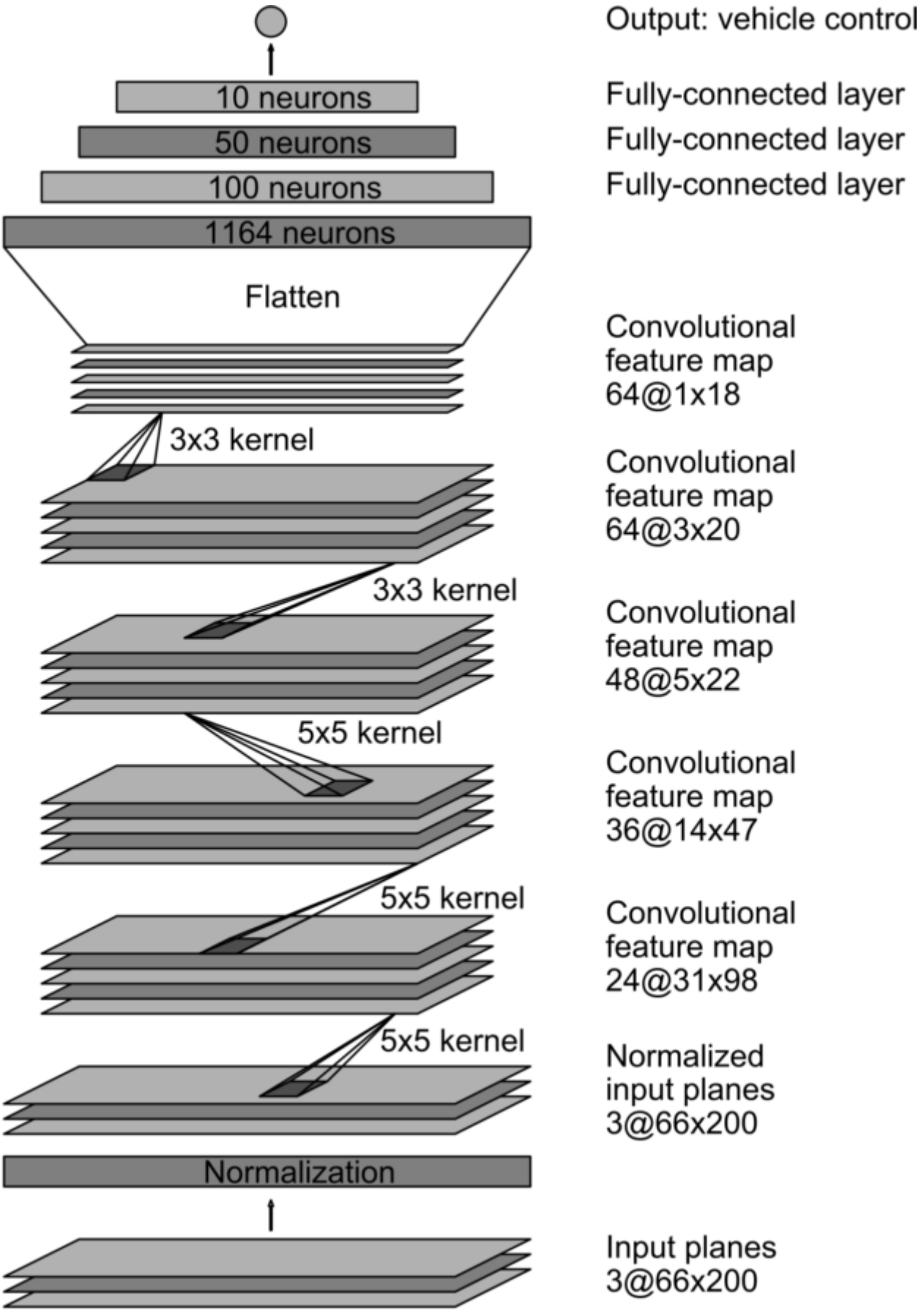
3. Submission code is usable and readable

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 multiple layers, the network architecture used is the NVIDIA network explained in the lecture videos.



The model RELU layers to introduce nonlinearity at each convolution step (code line 86,88,90,92,94,96), and the data is normalized in the model using a Keras lambda layer (code line 82).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 102).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 115-116). 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 115).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. My training data consisted of different driving sessions. I used the following combindations

- 2 -3 laps of normal driving
- 2-3 laps of reverse driving around the track
- Few laps of side to center driving
- Few laps of driving on the alternate track to make the driving behaviour more general

Model Architecture and Training Strategy

1. Solution Design Approach

I followed how the tutorial has explained the different models, starting from a simple convolution and its performance to a complex netwrok with an improved performance.

I also followed the video of improved perforamce that came with the NVIDIA network, as with most problems instead of starting from scratch I used the NVIDIA architecture as my basis.

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 an almost similar errors on the train and the validation sets at each epoch. To prevent overfitting I used a dropout layer.

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. To improve the driving behavior in these cases, I used the following approaches

- Generate more training data under different driving conditions in addition to the center lane driving.
- Augement the existing image, by flipping the training images (model.py line 52)

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) 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

conv2d_1 (Conv2D) (None, 31, 158, 24) 1824

conv2d_2 (Conv2D) (None, 14, 77, 36) 21636

conv2d_3 (Conv2D) (None, 5, 37, 48) 43248

conv2d_4 (Conv2D) (None, 3, 35, 64) 27712

conv2d_5 (Conv2D) (None, 1, 33, 64) 36928

flatten_1 (Flatten) (None, 2112) 0

dense_1 (Dense) (None, 100) 211300

activation_1 (Activation) (None, 100) 0

dropout_1 (Dropout) (None, 100) 0

dense_2 (Dense) (None, 50) 5050

activation_2 (Activation) (None, 50) 0

dense_3 (Dense) (None, 10) 510

activation_3 (Activation) (None, 10) 0

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

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to steer out from edges if it were to go over the edges. These images show what a recovery looks like



Then I repeated this process on track two in order to get more data points.

To augment the data set, I also flipped images and angles thinking that this would help to generate more data points for training and also generalize the driving behaviour. For example, here is an image that has then been flipped:


After the collection process, I had 25722 number of data points. I then preprocessed this data by normalizing it first(model.py line 82) and then cropping it (model.py line 84)

I finally 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. I used an epoch size of 5. I used an adam optimizer so that manually training the learning rate wasn't necessary.