Swapnil Daingade (sdaingad)

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:

- **clone.py** containing the script to create and train the model **(NOTE! My script is called clone.py and not model.py)**
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- **writeup_project4.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 **clone.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.

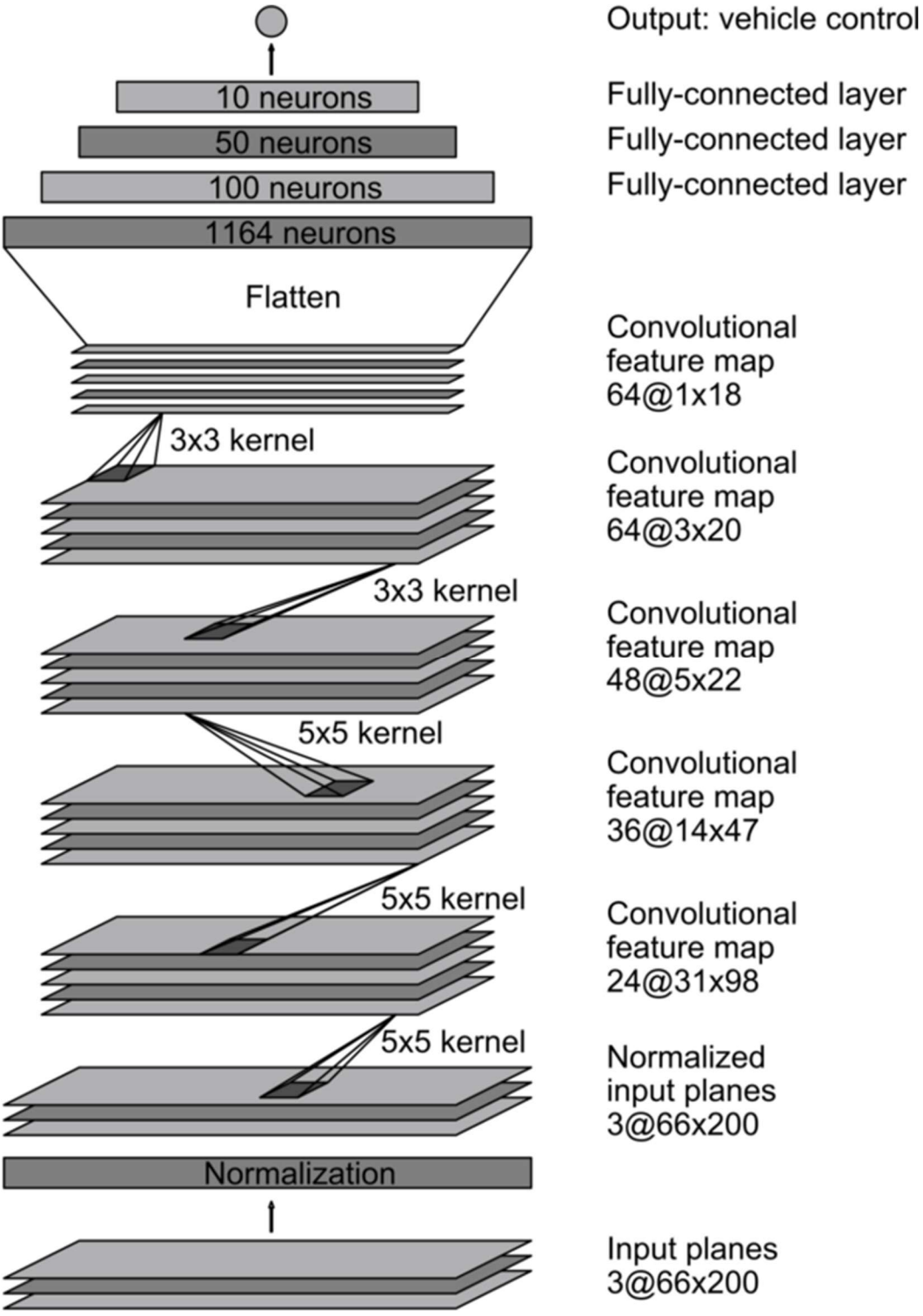
Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

I have implemented the model similar to the one described by the Nvidia paper “End-to-End Deep Learning for Self-Driving Cars”
here https://devblogs.nvidia.com/deep-learning-self-driving-cars/. It consists of

a. A normalization lambda layer
b. A cropping layer
c. 5 Convolutional layers with varying filter sizes and filter depths.
   Filter sizes for CNNs vary from (3x3 to 5x5). Filter depth for CNN layers varies from 24 to 64
d. Some of the convolutional layers have max pooling layers separating them to avoid overfitting
e. 4 Fully connected layers followed by the output layer

Here is the neural network as described in the Nvidia paper



Here is the one that I built in the **clone.py**. Output was generated by a call to model.summary()

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 90, 320, 3)	0
conv2d_1 (Conv2D)	(None, 86, 316, 24)	1824
max_pooling2d_1 (MaxPooling2	(None, 43, 158, 24)	0
conv2d_2 (Conv2D)	(None, 39, 154, 36)	21636
max_pooling2d_2 (MaxPooling2	(None, 19, 77, 36)	0
conv2d_3 (Conv2D)	(None, 15, 73, 48)	43248
conv2d_4 (Conv2D)	(None, 13, 71, 64)	27712
conv2d_5 (Conv2D)	(None, 11, 69, 64)	36928
flatten_1 (Flatten)	(None, 48576)	0
dense_1 (Dense)	(None, 1164)	56543628
dense_2 (Dense)	(None, 100)	116500
dense_3 (Dense)	(None, 50)	5050
dense_4 (Dense)	(None, 10)	510
dense_5 (Dense)	(None, 1)	11

Total params: 56,797,047
Trainable params: 56,797,047
Non-trainable params: 0

2. Attempts to reduce overfitting in the model

The model contains max pooling layers in order to reduce overfitting (clone.py lines 81, 83).

The dataset was split 80 – 20% for training and validation (clone.py line 64).

Both the training and validation losses came down after 5 epochs and were comparable. In case of an overfit model the training loss would be low but validation loss would be high. That was not seen here.

2019-06-21 10:02:28.895568: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1045] Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla K80, pci bus id: 0000:00:04.0)

201/201 [==============================] - 154s 768ms/step - loss: 5.7046 - val_loss: 0.0206

Epoch 2/5

201/201 [==============================] - 144s 718ms/step - loss: 0.0180 - val_loss: 0.0189

Epoch 3/5

201/201 [==============================] - 144s 718ms/step - loss: 0.0164 - val_loss: 0.0189

Epoch 4/5

201/201 [==============================] - 144s 718ms/step - loss: 0.0155 - val_loss: 0.0187

Epoch 5/5

201/201 [==============================] - 144s 718ms/step - loss: 0.0148 - val_loss: 0.0184

The data was shuffled to make sure the model did not memorize the steering angles.

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 (clone.py line 97).

4. Appropriate training data

The following processing was done for training data

a. Data was shuffled to avoid the model memorizing steering angles
b. Data was normalized and zero centered with the help of a Lambda layer
c. Images were cropped at top (50pixels) and bottom (20px) to focus on relevant information
d. Significant data augmentation was done. I used images from all three cameras Left, Center and Right.
e. Steering angles were adjusted with a correction of -/+ 0.2 for Left and Right images respectively.
f. In addition to using all three cameras, Images from all three cameras were flipped along the vertical axis to avoid left turn bias.
g. Steering angle was adjusted correctly for vertically flipped images

Interesting training was only done with center lane driving. No recovery driving was done from the left and right side of the road.
This actually puzzled me initially. How was it working ? **I think the reason the model is working correctly even without any recovery**
driving is because of the use of left and right images and the appropriate adjustment of steering angles. These function similar to recovery driving.
Also flipping Left, Center and Right images along the vertical axis along with appropriate adjustment of steering angle seems to have helped.

Model Architecture and Training Strategy

1. Solution Design Approach

I started with a neural network that Nvidia had used successfully in their paper instead of trying to engineer one from scratch.

This was inspired by the lecture videos that talked about using networks like VGG, ResNet, InceptionV3 etc to do object detection instead of engineering one from scratch.
This saved me significant time and effort.

After that my strategy was to improve the quality of the dataset by doing preprocessing and data augmentation.

The following processing was done for the training data

a. Data was shuffled to avoid the model memorizing steering angles
b. Data was normalized and zero centered with the help of a Lambda layer
c. Images were cropped at top (50pixels) and bottom (20px) to focus on relevant information
d. Significant data augmentation was done. I used images from all three cameras Left, Center and Right.
e. Steering angles were adjusted with a correction of -/+ 0.2 for Left and Right images respectively
f. In addition to using all three cameras, Images from all three cameras were flipped along the vertical axis to avoid left turn bias.
g. Steering angle was adjusted correctly for vertically flipped images

I used two rounds around the track of center lane driving to generate the dataset. Left and right camera images and steering angle adjustments helped compensate for lack of recovery driving as discussed above.
I was able to get low training and validation losses after 5 epochs suggesting that the model was not overfitting to the training data.

2. Final Model Architecture

The final model architecture (clone.py lines 76-92) consisted of a convolution neural network with the following layers and layer sizes ...

Output generated using model.summary() (clone.py line 95)

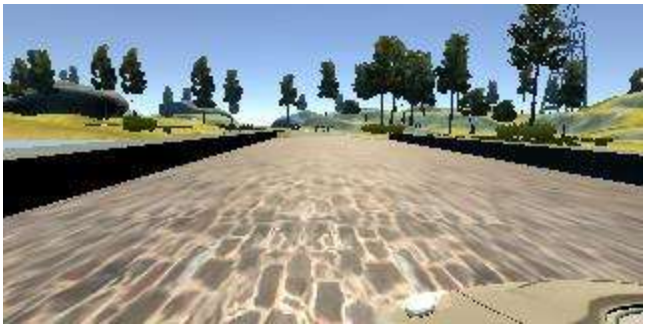
Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 90, 320, 3)	0
conv2d_1 (Conv2D)	(None, 86, 316, 24)	1824
max_pooling2d_1 (MaxPooling2	(None, 43, 158, 24)	0
conv2d_2 (Conv2D)	(None, 39, 154, 36)	21636
max_pooling2d_2 (MaxPooling2	(None, 19, 77, 36)	0
conv2d_3 (Conv2D)	(None, 15, 73, 48)	43248
conv2d_4 (Conv2D)	(None, 13, 71, 64)	27712
conv2d_5 (Conv2D)	(None, 11, 69, 64)	36928
flatten_1 (Flatten)	(None, 48576)	0
dense_1 (Dense)	(None, 1164)	56543628
dense_2 (Dense)	(None, 100)	116500
dense_3 (Dense)	(None, 50)	5050
dense_4 (Dense)	(None, 10)	510
dense_5 (Dense)	(None, 1)	11

Total params: 56,797,047
Trainable params: 56,797,047
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 used Left and Right camera images as well



I flipped all three images along the vertical axis. So for each data sample I had 6 images.

My driving_log.csv had 8036 entries. Each entry resulted in 6 images. Therefore total images were 48,212.

The data was split between training and validation in the ratio (80% to 20%)

Additional data processing was done as described above.

I ran training for 5 epochs and notice that the training and validation loss decreased to a lot value indicating there was low chance of overfitting.