Behavioral Cloning

Write-up Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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_nvidia.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 Beta for Mac 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_nvidia.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.

Code is commented

Model Architecture and Training Strategy

1. **An appropriate model architecture has been employed**

Figure below shows the network architecture, which consists of 9 layers, including a normalization layer, 5 convolutional layers, and 4 fully connected layers.

The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture, and to be accelerated via GPU processing.

The convolutional layers are designed to perform feature extraction, and are chosen empirically through a series of experiments that vary layer configurations. We then use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel, and a non-strided convolution with a 3×3 kernel size in the final two convolutional layers.

We follow the five convolutional layers with three fully connected layers, leading to a final output angle value for the steering.

My model consists of a convolution neural network with 3x3 filter sizes and depths between 32 and 128 (model_nvidia.py lines 82-98)

The model includes RELU layers to introduce nonlinearity (code line 86-91), and the data is normalized in the model using a Keras lambda layer (code line 82).

diagram credit https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets including CW and CCW direction. The more the data the lesser the chance for overfitting. 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 25).

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. When using the left and right cameras a correction (0.2) was introduced for the steering value for compensation. The value for 0.2 was recommended in one of the sessions by David.

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 make sure the car responded well in autonomous mode. The strategy involved three phases:

1. Phase 1: Create a simple on ConvNet based neutral network to make sure the plumbing was working
2. Phase 2: Since LeNet was very popular I chose to go with the same. I implemented the same as below.

```
model = Sequential()

model.add(Lambda(lambda x: (x/ 255.0) - 0.5, input_shape=(160,320,3)))

model.add(Cropping2D(cropping=((70,25), (0,0))))
```

```
model.add(Convolution2D(6,5,5, activation="relu"))

model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(120))

model.add(Dense(84))

model.add(Dense(1))
```

This involved one ConvNet with 5x5 with no striding. However, it didn't work well and the car often went into the water. The LeNet model didn't work and the car drowned in the spot below:

2. Final Model Architecture

Phase 3: In phase #3 I used a convolution neural network model similar to the one proposed from Nvidia and described by David. Since this model was highly talked about I went ahead to implement this rather than troubleshoot with one with LeNet. I also used image cropping to reduce distractions. which consists of 9 layers, including a normalization layer, 5 convolutional layers, and 4 fully connected layers.

The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture, and to be accelerated via GPU processing.

The convolutional layers are designed to perform feature extraction, and are chosen empirically through a series of experiments that vary layer configurations. We then use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel, and a non-strided convolution with a 3×3 kernel size in the final two convolutional layers.

We follow the five convolutional layers with three fully connected layers, leading to a final output angle value for the steering. I also deployed the fit_generator to minimize the resources required to run the model training. The model was trained on AWS/GPU and the model was downloaded back to my mac for driving using drive.py.

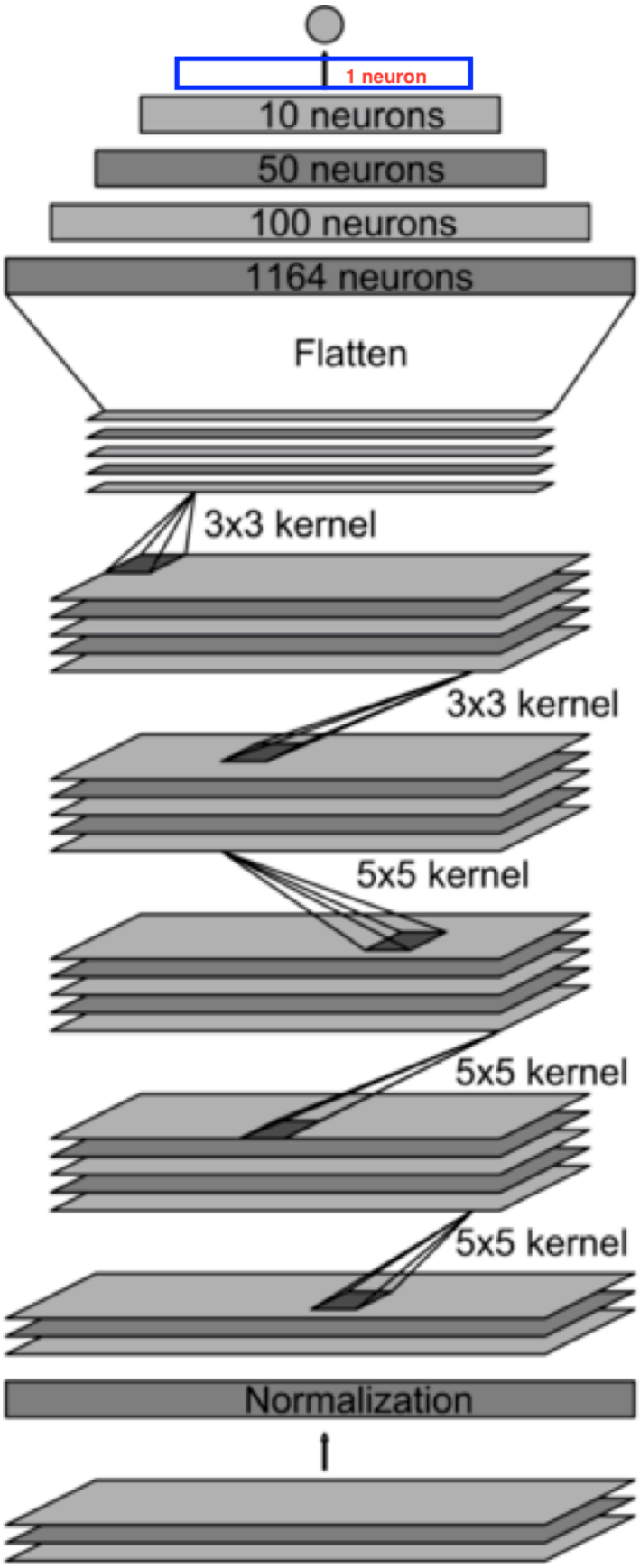
My model consists of a convolution neural network with 3x3 filter sizes and depths between 32 and 128 (model_nvidia.py lines 82-98)

The model includes RELU layers to introduce nonlinearity (code line 86-91), and the data is normalized in the model using a Keras lambda layer (code line 82).

// Below implemented based on review

Dropout was implemented with the value of 20% to reduce overfitting. This was based on reviewer's recommendation so that overfitting of data in minimized.

PS: I may have to work more on tuning the dropouts as the overall loss has not reduced and the simulator does not have signification improvement. (Actually, it degraded).



Output: vehicle control

Fully Connected layer

Fully-connected layer

Fully-connected layer

Fully-connected layer

Convolutional feature map 64@1x18

Convolutional feature map 64@3x20

Convolutional feature map 48@5x22

Convolutional feature map 36@14x47

Convolutional feature map 24@31x98

Normalized input planes 3@66x200

Input planes 3@66x200

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set using keras instructions. Magically the NVIDIA model worked really well off the shelf. I used 7 epochs maybe a bit more than needed but wanted to make sure the loss was minimized. The same oscillated a bit towards the end but it was low enough.

```
Epoch 1/7
8064/7976 [==============================] - 21s - loss: 0.0393 - val_loss:
0.0299
Epoch 2/7
8064/7976 [==============================] - 16s - loss: 0.0396 - val_loss:
0.0439
Epoch 3/7
8064/7976 [==============================] - 16s - loss: 0.0318 - val_loss:
0.0386
Epoch 4/7
8064/7976 [==============================] - 16s - loss: 0.0366 - val_loss:
0.0287
Epoch 5/7
8064/7976 [==============================] - 16s - loss: 0.0323 - val_loss:
0.0219
Epoch 6/7
8112/7976 [==============================] - 17s - loss: 0.0302 - val_loss:
0.0301
Epoch 7/7
8064/7976 [==============================] - 15s - loss: 0.0288 - val_loss:
0.0417
```

The final step was to run the simulator to see how well the car was driving around track one. The NVIDIA model based on the 9 layers worked well and needed no retraining or adjustment at all. This model was also trained on the CW and CCW data. Please note the mac sim beta was used for training and autonomous testing.

https://d17h27t6h515a5.cloudfront.net/topher/2017/February/5894ecbd_beta-simulator-mac/beta-simulator-mac.zip

At the end of the process, the vehicle could drive autonomously around the track without leaving the road. This is shown in the video file NVidiaRun1.mp4

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

As we can see from the visualization that there is oscillation and the 6th epoch was higher. The seventh was back to the minimal. Also overall the MSE was quite low and therefore I ignored the oscillation.

Please note since I trained on GPU/AWS, also the graph below is from my Mac with no GPU. The matlibplot only worked on my Mac therefore there are two sets of MSE (one in green on AWS and the one in blue on my local Mac).

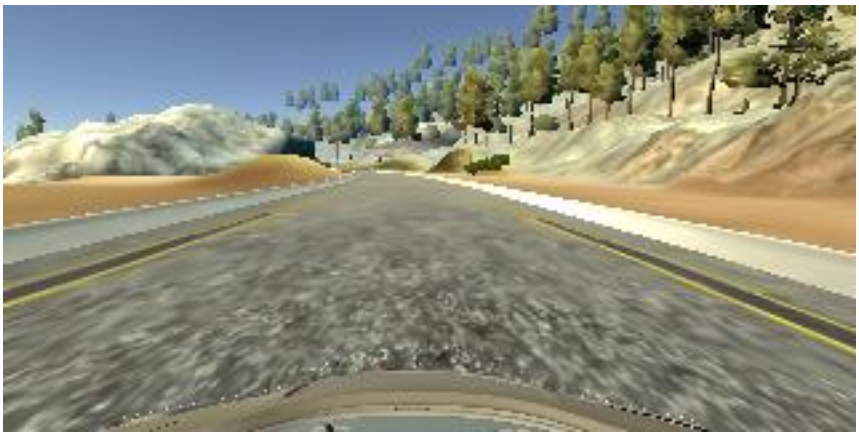
below graph is based on second iteration that includes dropouts.


```
python model_nvidia.py
Using TensorFlow backend.
Epoch 1/7
4224/4128 [==============================] - 72s - loss: 0.0379 - val_loss:
0.0397
Epoch 2/7
4224/4128 [==============================] - 69s - loss: 0.0362 - val_loss:
0.0342
Epoch 3/7
4224/4128 [==============================] - 69s - loss: 0.0278 - val_loss:
0.0246
Epoch 4/7
4224/4128 [==============================] - 67s - loss: 0.0356 - val_loss:
0.0287
Epoch 5/7
4224/4128 [==============================] - 66s - loss: 0.0306 - val_loss:
0.0210
Epoch 6/7
4224/4128 [==============================] - 67s - loss: 0.0212 - val_loss:
0.0364
Epoch 7/7
```

```
4224/4128 [==============================] - 73s - loss: 0.0246 - val_loss:
0.0203
```

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 come back to the middle. Please note this was done in the bridge area only as shown below.

Then I repeated this process on same track in CW direction to get more data points and remove left bias.

After the collection process, I had 15480 number of data points. I then preprocessed this data by normalizing and cropping the same.

I finally randomly shuffled the data set and put of the data into a validation set. Line #70

```
train_samples, validation_samples = train_test_split(lines, test_size=0.2)
```

I used this training data for training the model.

Track Two

The simulator contains two tracks. To meet specifications, the car must successfully drive around track one. Track two is more difficult. See if you can get the car to stay on the road for track two as well.

Video file submitted Track2.mp4 along with modified script model_nvidia_T2.py.
However there is more work needed as its running into the hill ☺

Summary of Rubrics

PROJECT SPECIFICATION

Use Deep Learning to Clone Driving Behavior

Required Files

RITERIA	MEETS SPECIFICATIONS
ıre all required files submitted?	The submission includes a model.py file, drive.py, model.h5 a writeup report and

Quality of Code

CRITERIA	MEETS SPECIFICATIONS
Is the code functional?	Done The model provided can be used to successfully operate the simulat
Is the code usable and readable?	Done

CRITERIA	MEETS SPECIFICATIONS
	The code in `model.py` uses a Python generator, if needed, to gener data in memory. The `model.py` code is clearly organized and comm

Model Architecture and Training Strategy

CRITERIA	MEETS SPECIFICATIONS
Has an appropriate model architecture been employed for the task?	Done The neural network uses convolution layers with appropriate filter s model. The data is normalized in the model.
Has an attempt been made to reduce overfitting of the model?	Done Train/validation/test splits have been used, and the model uses drop
Have the model parameters been tuned appropriately?	Done Learning rate parameters are chosen with explanation, or an Adam
Is the training data chosen appropriately?	Done Training data has been chosen to induce the desired behavior in the

Architecture and Training Documentation

CRITERIA	MEETS SPECIFICATIONS
Is the solution design documented?	Done The README thoroughly discusses the approach taken for derivin the given problem.

CRITERIA	MEETS SPECIFICATIONS
Is the model architecture documented?	Done The README provides sufficient details of the characteristics and model used, the number of layers, the size of each layer. Visualizat architecture are encouraged.
Is the creation of the training dataset and training process documented?	Done The README describes how the model was trained and what the (as how the dataset was generated and examples of images from the

Simulation

CRITERIA	MEETS SPECIFICATIONS
Is the car able to navigate correctly on test data?	Done No tire may leave the drivable portion of the track surface. The car surfaces that would otherwise be considered unsafe (if humans wer