CARND-TERM 1 Project 3: Behavioural Cloning Project

Rubric Point

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. ipynb 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/model_LeNet_best.h5
```

The other .h5 files are for debugging and reference purposes.

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

I have implemented the NVIDIA architecture first, which consists of a convolution neural network with 5x5 and 3x3 kernels, before finally settling on simpler and seemingly more consistent architecture based LeNet used in Project 2.

The model includes RELU layers to introduce nonlinearity and the data was normalized using a Keras lambda layer.

2. Attempts to reduce overfitting in the model

The model contains dropout layers and max pooling layers to reduce overfitting (line 193, 188, 191 etc.)

The model was trained and validated on different data sets to ensure that the model was not overfitting (line 168). 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, therefore the learning rate was not tuned manually (line 195).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of centre- lane driving, recovery driving etc.

For details about how I created the training data, see the next section.

Architecture and Training Documentation

1. Solution Design Approach

In order to verify whether the setup environment could work properly, I deployed the simpliest CNN with just 1 convolution layer and a single fully connected layer. The test environment was unable to launch initially but the issue was fixed after I reinstalled socketio and eventlet modules using Anaconda Prompt. At this stage, the model was still outrageously immature and it was no surprise that my car veered straight off the track to the left and sank silently to the bottom of the lake right after the autonomous mode on simulator was activated, incredibly, the engine did not appear to be damaged in any way (waterproof engine? That would be wonderful in real life!) and the car was travelling in a rather haphazard way with no luck of getting out the lake anytime soon. To be honest, I was laughing all the way to my bed that night as I kind of made my first autonomous submarine! Having been satisfied with the environment setup, it was finally time for making the all-important call on choices of deep learning CNN architectures.

The overall strategy for deriving a model architecture was to trial several contrasting architectures involving a variety of convolution and FC layer combinations.

Initially, I adopted the NVIDIA model straightaway, however, it proved to be gruelling to tame as I struggle to keep the car on the track. I did have some limited success with NVIDIA model though, by not incorporating recovery data for Track one nor data from Track Two, the car only just managed to stay within the drivable portion of the track one. The result from NVIDIA model are recorded as a mp4 file, navigate to MP4/Nvidia_best.mp4 to see how the car completed two laps on track one.

In addition to that, I reused the LeNet architecture from project 2, modified it slightly, to my surprise, the performance yielded from this model was noticeably more consistent as the car stayed within the track limit at all time (please navigate to MP4/LeNet_FINAL.mp4 to see the result) and 2 CONV layers + 2 FC layers were all I needed to create a satisfactory model, giving a much more consistent performance than NVIDIA model in this case, which was very puzzling indeed.

To gauge how well the model was working, I split my image and steering angle data into a training and validation set with a 9:1 ratio.

To combat the overfitting, I added max pooling and dropout regularisation.

For NVIDIA model: The car could complete 5 clean laps, but exceeded the track limit (defined as one or more wheels off the yellow lines on both sides of the track or riding the red-white kerbs) several times at the twisty section after the bridge, the direction changes were sometimes sudden and hard.

For LeNet model: The car could complete 5 clean laps without ever exceeding the track limit, the direction changes were mostly smooth and stable.

2. Final Model Architecture(LeNet)

The final model architecture based on LeNet consisted of a convolution neural network with the following layers and layer sizes:

- Layer 1: 5x5 Filter with depth 16, stride = 1x1
- MaxPooling (2x2)
- Layer 2: 5x5 Filter with depth 32, stride = 1x1
- MaxPooling (2x2)
- Flatten
- Fully Connected Layer A : n = 32 Dropout Layer : Dropout Value = 0.5
- Fully Connected Layer B : n = 1

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
cropping2d_1 (Cropping2D)    (None, 65, 320, 3)        0
_________________________________________________________________
lambda_1 (Lambda)            (None, 65, 320, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 316, 16)       1216
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 158, 16)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 154, 32)       12832
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 77, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 32032)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1025056
_________________________________________________________________
activation_1 (Activation)    (None, 32)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,039,137
Trainable params: 1,039,137
Non-trainable params: 0
```

3. Creation of the Training Set & Training Process

The simulator (not 64-bit) was launched at a resolution of 1024x768 and the quality was set to 'fantastic' as it was clear to me that shadows could play a huge role.

Recording my driving behaviour was not a walk in the park, I had to constantly monitor the positions and velocities of the car while on track.

Initially, I decided to record training data while driving at the maximum speed allowed by the simulator of 30MPH throughout the recording session I violated several crucial traffic regulations by not slowing down at all while taking corners.

My first training set consists of 38694 set of data, which combines all data in a single package (Track A, Track B, recovery, centre-lane driving etc.) I was sizzling with confidence that the model would work. Unfortunately, the sight of the car unable to take the sharpest corners on Track one forced me to rein in my complacency. That was before I decided to slow down the car while recording training data, maximum speed was then capped at around 8MPH in all subsequent recording sessions.

To be more organised with my data, I recorded 4 more separate training-data packages:

Data Package Detail	**Sample Size**	**Used in the final model?**	**Lap Details**
DrivingDataTrackA	12314	YES	2 clockwise + 2 anticlockwise
DrivingDataTrackA_enhanced	4746	YES	Driving back and forth on the bridge and twisty section of the track
DrivingDataTrackA_Recovery	5555	NO	Recovery driving from the side of the track at a number of sharp corners and the bridge
DrivingDataTrackB	15740	NO	All the above for track B
DrivingData (primary)	38694	NO	All the above

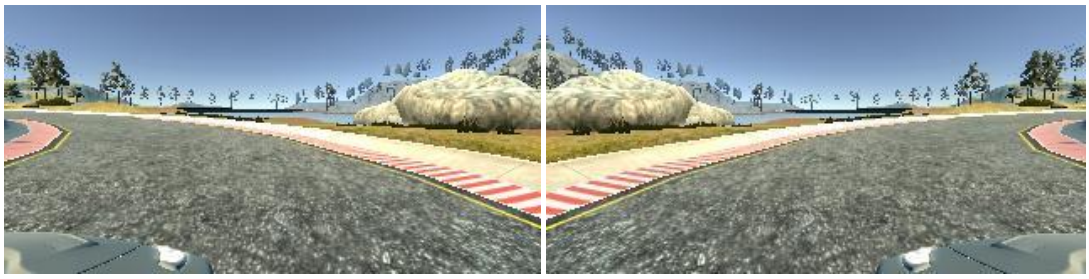
However, as I mentioned before, only DrivingDataTrackA, DrivingDataTrackA_enhanced were used as the training data, the addition of Track-two and recovery data seemed to severely degrade the model performance on Track one.

Some example images:

Centre-lane driving:

Recovery Driving:

Augmentation by flipping images:



After the collection process, I had 17060 number of data points. I then pre-processed this data using normalisation.

I finally randomly shuffled the data set and put 10% 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 epoch number epoch number and batch size was set to 5 and 32 respectively as they strike a good balance between speed and accuracy. The Adam Optimizer meant that manually training the learning rate wasn't necessary.

References:

www.keras.io

www.nvidia.com

www.github.com