Write Up for P4: Behavior Cloning

Files submitted

Files list

This document is a write up for the 4th project. My submission includes the following files:
Git repository: https://github.com/mmarouen/CarND-Behavioral-Cloning-P3

- **Model.py** code for building and saving the model
- **Model2.h5** 2nd track model object
- **WriteUp_P4_Behavior_Cloning.pdf** Writeup, this file
- **Model_generator.py** includes a generator implementation. Not used because very slow, I keep it for reference.

Google drive: https://drive.google.com/open?id=1uAiaHx4QdtSlq7Rlr3NCusLo0iENRhwR

- **Model1.h5** 1st track model object (size > 25 mb)

Functional code

I wrote and trained the models on the environment provided by Udacity. For the first track, I trained using only the provided data (+ augmentations). So the first track model should duplicable.

I used the following folder structure:

```
--CarND-Behavior-Cloning-P3
  |--model.py
  |--drive.py
  |--video.py
--Data
  |--IMG
  |--driving_log.csv
--Data2
  |--IMG
  |--driving_log.csv
```

NB: ***Data*** and ***Data2*** contain data for track 1 and track 2 respectively. I could not mix both because of memory problems.

NB: Images URL in *driving_log.csv* must start with ***IMG/...*** not with absolute path

NB: URL for the data folder must be set in `model.py` via *"file_url" line 14*

```
13
14  file_url='../data/'
```

⇒ To train the model: `python model.py`

NB: Please set epochs count as follows **epochs=10** for first track, **20** for the second track

```
98  model.fit(Xtrain,ytrain,batch_size=48,shuffle=True,validation_split=0.2,epochs=10)
```

⇒ To drive the car on the 1st track: `python drive.py model.h5`

⇒ To drive the car on the 2nd track: `python drive.py model2.h5`

Model architecture and training strategy

Architecture

I used a model architecture inspired by AlexNet. The main ideas are:
-Increase features depth + **decrease** image size, filter shape
-Decrease image size by Maxpooling, Conv layer does not alter size
I came to this architecture as follows:I started initially with 2 Conv layers
I noticed that training error was not optimal.
⇒ So I started densifying the network by increasing features and adding some CONV layers.
⇒ Batch normalization + grayscale help the model generalize
Concerning nvidia structure:
I compared with the suggested nvidia architecture but I didn't notice an improvement. My explanation is that data collection process also plays an important role. I did not find enough information about that. So I prefer to keep below architecture

Name	Layer	Description	Output Shape
	Input	160x320x3	-
Preprocessing block	Scaling	Scale in [-1 .. 1]	N_train x 160x320x3
	Convert to Gray	Convert images to grayscale	N_train x 160x320x1
	Cropping	Removes upper and bottom rows	N_train x 70x320x1
CONV 1	CONV 5x5	Stride: 1x1, same padding, filter depth: 16	N_train x 70x320x16
	BatchNorm → ReLU → Dropout 0.5		N_train x 70x320x16
	Max POOL 2x2	Stride: 2x2	N_train x 36x180x16
CONV 2	CONV 5x5	Stride: 1x1, same padding, filter depth: 32	N_train x 36x180x32
	BatchNorm → ReLU → Dropout 0.5		N_train x 36x180x32
	Max POOL 2x2	Stride: 2x2	N_train x 18x180x32
CONV 3	CONV 3x3	Stride: 1x1, same padding, filter depth: 64	N_train x 18x180x64
	BatchNorm → ReLU → Dropout 0.5		N_train x 18x180x64
	Max POOL 2x2	Stride: 2x2	N_train x 9x90x64
CONV 4	CONV 3x3	Stride: 1x1, same padding, filter depth: 64	N_train x 9x90x64
	BatchNorm → ReLU → Dropout 0.5		N_train x 9x90x64
	Max POOL 2x2	Stride: 2x2	N_trainx 4x45x64
FC1	Sigmoid → BN → ReLU → Dropout 0.5		N_train x 128
FC2	Sigmoid → BN → ReLU → Dropout 0.5		N_train x 64
FC3	Final layer		N_train x 1

Reduce overfitting & parameters tuning

I try to follow the following process to reduce overfitting and to increase generalization capability of the model:

1. Minimize training error independently from validation error:
    a. Increase features volume
    b. Increase layers count
    c. Include batch normalization
    d. Scale data in [-1 .. 1]
    e. Convert to grayscale if the colors are not very relevant
    f. Use efficient backprop techniques such as 'adam' optimizer

    ```
    model.compile(optimizer='adam',loss='mse')
    ```

    g. Have more data
    h. Increase number of epochs

    ⇒ Eventually I reach a minimum error. That error becomes my target for the validation error
2. Minimize validation error to reach target error:
    a. Gradually adding dropout layers + reduce keep_prob
    b. Include batch normalization
    c. Change mini batch size
    d. Get more data
    e. Increase number of epochs

Data creation

Track 1

I only used the provided data.

Track 2

I collected 11,123 frames corresponding to approximately 4 laps.

```
root@865a7e90f3b0:/home/workspace# wc -l data2/driving_log.csv
11123 data2/driving_log.csv
```

I try to keep the car centered in the lanes as follows:



I also include examples of recovering from left (read left → right)

Recovering from right (read left → right)

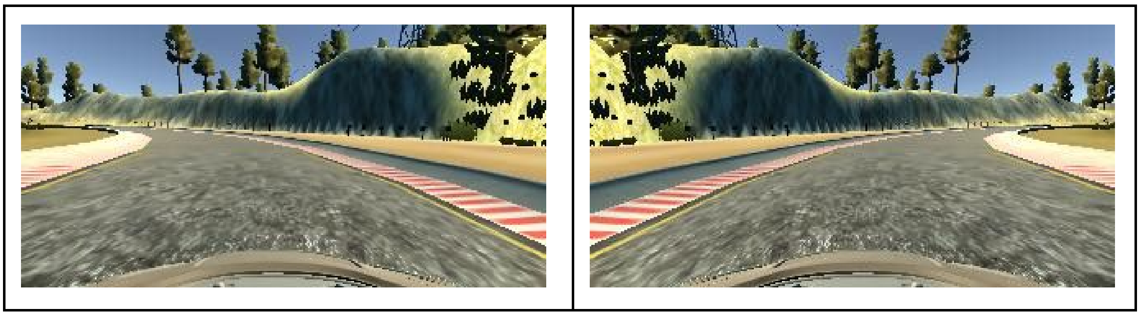
Data augmentation

The process of data augmentation is the same for both tracks. I follow the recommended approach presented in the course. It multiplies the count of total data by a factor of 6:

1. Use data from 3 cameras: Here’s an example of the same frame from 3 cameras

Center	Left	Right
		

2. Flip each frame vertically



Data process

Once data augmentation is finished, we endup with the following data quantity:

	Track 1	Track 2
Total raw	8,037	11,123
Total augmented	48,222	66,738
Training (**80%**)	38,577	53,390
Validation (**20%**)	9,645	13,348

Discussion

This project was beneficial for me to apply neural network modeling on Keras and using car driving data, I think that this is a very interesting E2E application.
I have 2 remarks:

- I believe that it's better to split the validation data **before** augmentation. That way, we make sure that validation data is exactly the real distribution.
- I tried generators but it was very slow even after tweaking some parameters of ***fit_generator*** function. One epoch takes around 5h !