Self-driving car Nanodegree (Term 1). Project 3: Behavioral Cloning

Ruggero Vasile

June 28, 2017

Abstract

Report for the third project in the self-driving car nanodegree program, term 1. In this project the aim is to teach the vehicle of the simulator to drive on the simple track example indefinitely.

1 Introduction

Behavioural cloning consists in teaching an agent to perform correct actions using previous recorded actions by another agent, in this case a human agent.

In this project we are required to teach a car to drive on a closed track exploiting the behaviour of human agents. The human agent is required to record successful driving laps on a simulator and to use this data to make the agent learn from his/her experience. Specifically the system will have to be able to estimate the correct steering angle at each single instant using only visual sensors, i.e. images coming from cameras installed on the front of the car. This problem therefore can be casted as a regression problem where the output variable is the steering angle value and the input are three channel images.

While more advanced computer vision algorithms could be used to help the system to learn the task better, in this project we use a direct deep learning approach without specific suggestion for the system behaviour. Our degrees of freedom are restricted to the choice of the convolutional network architecture to use and on some sort of preprocessing of the data used as input. In the next sections both these two aspects will be explored.

2 The dataset

The dataset used for training is the one provided by Udacity. It was suggested to try to collect the data on our own using clever strategies, especially regarding what kind of driving techniques to use. This last approach was almost successful but did not allow me to train a model leading to a smooth driving agent. Therefore I present here the results obtained using the Udacity dataset.

The set consists of a collection of images captured at different instant of time. For each instant the data contains three different images taken from a central camera and two cameras slightly shifted on the left and right of the central camera. Moreover the values of the steering angle, throttle, brake and speed are also captured. Apart from the steering angle the other quantities are of no particular interest to this project since we will fix the simulator speed during testing to a fixed value (10). The steering angle takes values in the interval $[-1, 1]$ corresponding to $[-25deg, +25deg]$ of the simulator interface. In Fig. 1 I show an example of camera images taken at a given instant of time from the three views. All these images are in RGB color space and their size is 160×320 pixels.

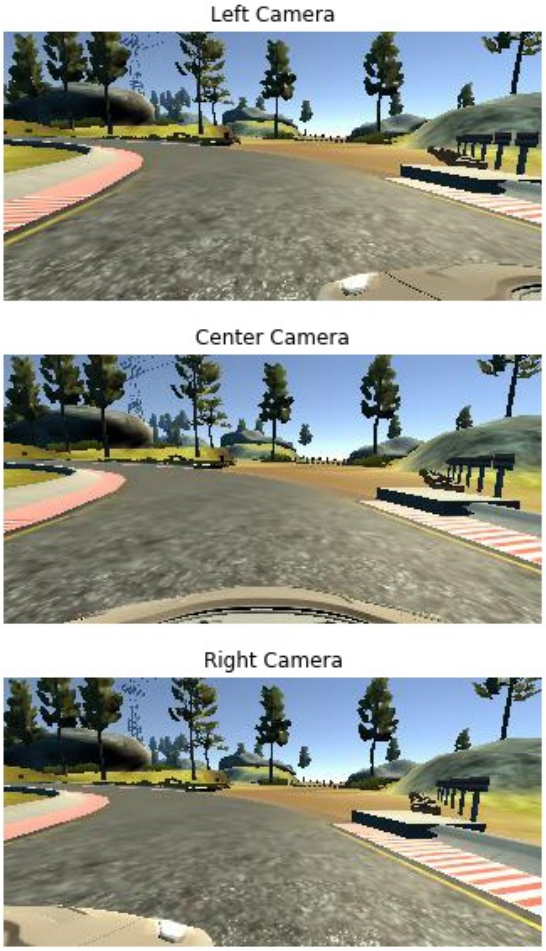

Figure 1. Left, center and right images taken at the same instant of time.

2.1 Data preprocessing and augmentation

Before feeding the data to the model I impose a data preprocessing pipeline. The model used is the same as the one suggested by NVIDIA (http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf), which require an input image of 66×200. In order to transform the images into this format I apply the following pipeline:

- Resizing the image having width of 200 keeping aspect ratio.
- Cropping the image from above until the height is equal to 66. The image are cropped from above since the sky part of the image is not contributing to the result.

As suggested in the NVIDIA reference it is also possible to transform the image into the YUV color space. However this was not an important requirement for the model to work properly. Since deep learning approaches require a large amount of data, it is important to apply some kind of data augmentation. This is done on the fly since we use a generator function to prepare the training batches. Here the pipeline for a single sample:

- An image from the center, right or left camera is selected at random.
- The image is preprocessed
- The image is flipped along the vertical with 50% probability
- The image is shifted horizontally for a number of pixels extracted from a uniform distribution in the interval $[-25, 25]$.
- The steering angle for the specific example is corrected for: 1) The type of camera (0.25 for the left camera, 0 for the center camera and -0.25 for the right camera), 2) For the horizontal shift (see code *random_translation* function)

In Fig. 2 an example of preprocessing plus augmentation pipeline in action.

3 The model

The model chosen is the one suggested by NVIDIA. The specifics are in Table 1. It is important to notice that a further preprocessing is applied in the model via normalization that forces the pixels to have intensity values between -0.5 and 0.5. Dropout is also strongly imposed to avoid overfitting and the last layer has a linear activation with a single output since we are looking at a regression problem.

Training is done for 20 epochs, using batches of 64 samples. Adam optimizer with initial learning rate equal to 0.001 is used. I also use a validation set to monitor overfitting. This set is the whole training set of center camera images without data augmentation, i.e. without image flipping or shifting. Training takes in my machine about 1.5 hours while would take only 10 minutes on the AWS instance with GPU. I do not report here training error and validation since it is difficult to evaluate a specific limit of acceptance. The real test of satisfaction is the simulator.

4 Results

The video of simulator result with the trained model is attached to the project submission.

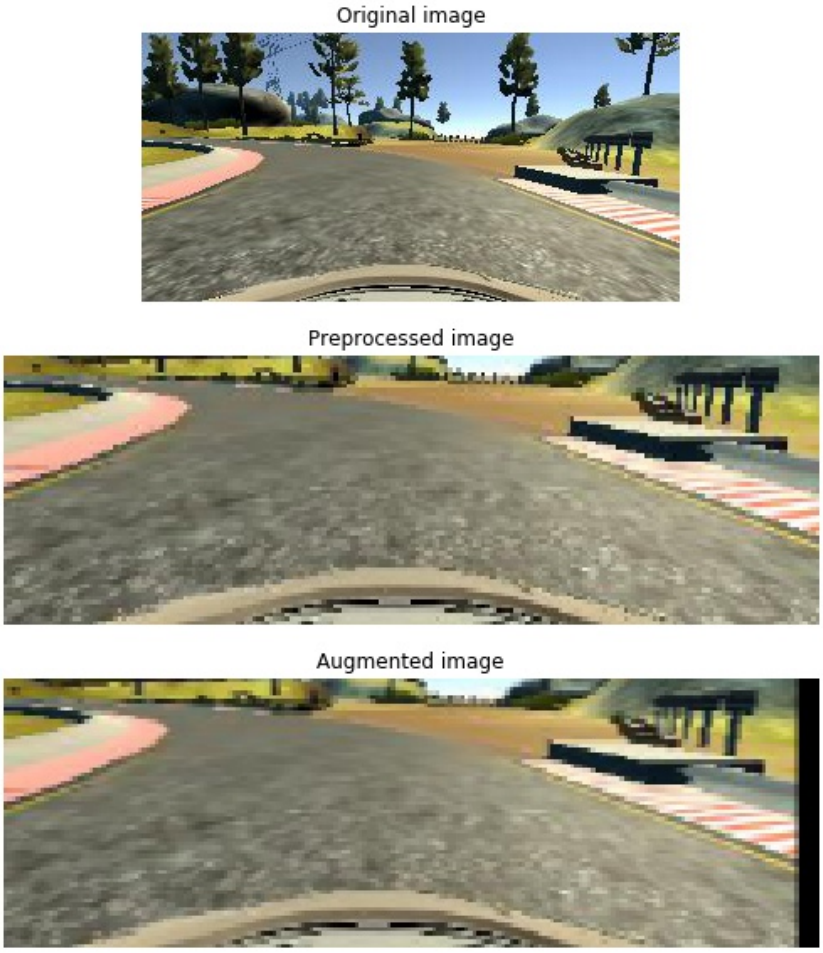

Figure 2. (Above) The original image. (Center) The image after resizing and cropping. (Below) The image after being shifted horizontally.

Layer	Input	Output	Other Hyperparameters
Normalization	66x200x3	66x200x3	x = x/255 - 0.5
Convolution + relu	66x200x3	31x98x24	kernel=5x5, sub=(2,2), pad=VALID
Convolution + relu	31x98x24	14x47x36	kernel=5x5, sub=(2,2), pad=VALID
Convolution + relu	14x47x36	5x22x48	kernel=5x5, sub=(2,2), pad=VALID
Convolution + relu	5x22x48	3x20x64	kernel=3x3, sub=(1,1), pad=VALID
Convolution + relu	3x20x64	1x18x64	kernel=3x3, sub=(1,1), pad=VALID
dropout			p_dropout=0.3
Fully Connected + relu	Flatten(1x18x64)	100	
dropout			p_dropout=0.3
Fully Connected + relu	100	50	
dropout			p_dropout=0.3
Fully Connected + relu	50	10	
dropout			p_dropout=0.3
Fully Connected	10	1	

Table 1. The final architecture utilized.